

Using M68HC908 ROM-Resident Routines

By **Rogelio Gonzalez Coppel**
RTAC Americas
Mexico

Overview

This document is a quick reference for using the ROM-resident routines to read, program, and verify the FLASH on any MC68HC908 MCU. Basic information about the functional description and configuration options will give the user a better understanding of how the ROM-resident routines work. This application note provides examples that demonstrate one use of the ROM-resident routines within the M68HC908 Family of microcontrollers. The examples given may be modified to suit requirements of a specific application.

This application note has a companion software file, AN2874SW.zip, available from www.freescale.com. It contains two example projects that demonstrate the routines described in this document.

This product incorporates SuperFlash® technology licensed from SST.

© Freescale Semiconductor, Inc., 2006. All rights reserved.

Introduction

Most M68HC908 MCUs have on-chip support routines used to program, erase, and verify the FLASH. These routines may be accessed in either user mode or monitor mode¹ and eliminate the need to develop separate FLASH routines for applications, saving development time and getting more robust FLASH performance.

There are two instances of the ROM-resident routines implementation in the HC908 MCUs, care should be taken to ensure the correct implementation is used (data structure type, arguments, routine addresses, etc.). The instance implemented on each device depends on the family of the MCU. These differences must be considered to correctly reference these routines.

This document will show how to use these routines from the user software and highlight the main differences between the two implementations.

Data Structure

Some routines require certain registers and/or RAM locations to be initialized before being called from the user software (via a JSR). The parameters passed to a routine are in the form of a contiguous data block stored in RAM. The index register (H:X) acts as a pointer to the routine and is loaded with an address that will depend on the instance implemented in the MCU. A data block has the control and data bytes in a defined order, as shown in Figure 1.

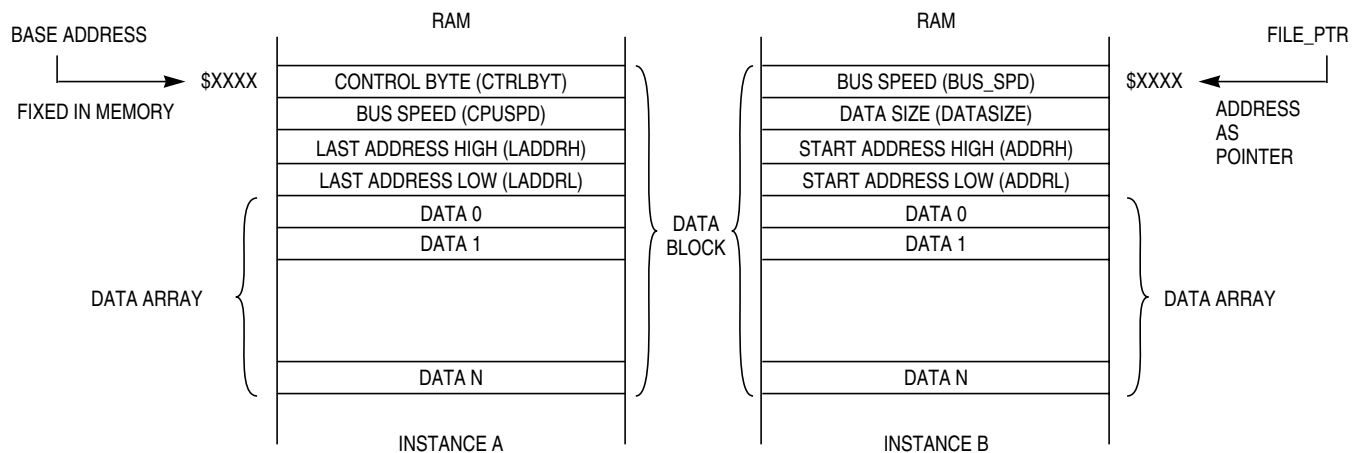


Figure 1. Data Block Format for ROM-Resident Routines

Instance A and instance B are similar, but there are differences in how the collection of variables is ordered in RAM. The next sections will describe in detail each implementation and which families use which instance.

1. User mode is the normal operating state of the MCU. Entering monitor mode requires specific conditions on pins. Monitor mode provides a programming interface with a PC via a standard RS-232 interface. For more on this, refer to the monitor ROM section in the MCU data sheet.

Instance A: Fixed Data Structure Implementation

Variables

Table 1 shows variables used in the ROM-resident routines and their locations. These variables are either passed in a register or as static variables in a predefined location in RAM (base address). This address will be located at the start of RAM + 8.

Table 1. Variables Used in ROM-Resident Routines — Instance A (See Notes)

Variable Name	Description	Location ⁽¹⁾
FADDR	Registers H and X are initialized with a 16-bit value representing the first address of a range.	H:X
CTRLBYT	Control byte used for the ERARNGE routine. Bit 6 in this location is used to specify either MASS (1) or PAGE (0) erase.	Base address
CPUSPD	CPU speed (used for delays within routines) is the internal operating frequency (f_{op}) in MHz times 4 then rounded up to the next integer ⁽²⁾ .	Base address + 1
LADDR	A range specifies the FLASH locations to be read, verified, or programmed. This 16-bit value in RAM holds the last address of a range.	Base address + 2
DATA	The DATA array contains data that is to be manipulated. The size of this array must match the size of the range to be programmed or verified.	Base address + 4

1. Base address is the fixed location in RAM at which the data array should be placed.

2. CPUSPD will be internal operating frequency (f_{op}) times 2 for M68HC908GZ8/16 and M68HC908GR16

ROM-Resident Routines

Table 2 introduces the routines. Details are discussed later in this document.

Table 2. ROM-Resident Routines — Instance A

Routine Name	Description
PRGRNGE	PRGRNGE is used to program ⁽¹⁾ a range of FLASH locations with data loaded into the DATA array.
ERARNGE	ERARNGE can be called to erase a page (64 bytes) or a whole array of FLASH.
RDVRRNG	This routine is used to perform one of two options. Using the send-out option, this routine reads FLASH locations and sends the data out serially on a general-purpose I/O. Using the verify option, this routine verifies the FLASH data against data in a specific RAM location, which is referred to as a DATA array.
GETBYTE	This routine is used to receive a byte serially on the general-purpose I/O port A bit 0. The receiving baud rate is the same as the baud rate used in monitor mode.
DELNUS	This routine can generate a specified delay based on the values of register X and accumulator (A) as parameters.

1. FLASH locations must be blank so they can be written to. User must ensure that the range specified is first erased before attempting to program it.

Addresses of Routines

Table 3 provides necessary addresses used in the on-chip FLASH routines for the HC908 families that use this instance. The address to call each of the five routines varies among the devices. This table gives the absolute address that should be used when calling the routines.

Table 3. MCU Type vs. On-Chip FLASH Routines Addresses (Instance A)

MCU	ROM-Resident Routines					RAM Data Base Address
	PRGRNGE	ERARNGE	RDVRRNG	GETBYTE	DELNUS	
EY16	\$1009	\$1006	\$1003	\$1000	\$100C	\$48
GR4/8	\$1CEC	\$1DA0	\$1CAD	\$1C99	\$1D96	\$48
GT8/16	\$1B59	\$1B56	\$1B53	\$1C6C	\$1B5C	\$48
JB8	\$FC09	\$FC06	\$FC03	\$FC00	\$FC0C	\$48
JB12/16	\$FC09	\$FC06	\$FC03	-	-	\$88
JG16	\$FC09	\$FC06	\$FC03	-	-	\$88
JK1/JK1E	\$FC09	\$FC06	\$FC03	\$FC00	\$FC0C	\$88
JL3/JK3	\$FC09	\$FC06	\$FC03	\$FC00	\$FC0C	\$88
KX2/8	\$1009	\$1006	\$1003	\$1000	\$100C	\$48
QF/QT/QY	\$2809	\$2806	\$2803	\$2800	\$280C	\$88
RK2/RF2	\$F04B	\$F07D	\$F00F	\$F2DE	\$F0EC	\$88

Note: Refer to AN1831 for a more detailed description on using these routines.

GZ8/16	\$1C09	\$1C06	\$1C03	\$1C00	\$1C0C	\$48
GR16	\$1C09	\$1C06	\$1C03	\$1C00	\$1C0C	\$48

Note: Refer to AN2545 for a more detailed description on using these routines.

LB8	\$038A	\$0387	\$0384	\$037E	\$038D	\$88
QL4	\$2B8A	\$2B87	\$2B84	\$2B7E	\$2B8D	\$88
QYxA/QTxA	\$2809	\$2806	\$2803	\$2800	\$280C	\$88
QB4/8	\$2809	\$2806	\$2803	\$2800	\$280C	\$88

Note: Refer to AN2635 for a more detailed description on using these routines.

Page Erase Issue

ERARNGE works properly when the mass erase operation is performed. However, we found that ERARNGE does not always fully erase a selected page when a page erase operation is performed. Furthermore, it has the potential to erase a vector page unintentionally.

Please refer to AN1831, Rev. 3, for more information on this issue, a list of affected devices, and workarounds.

Instance B: User-Defined Data Structure Implementation

Variables

For this implementation, the index register (H:X) is loaded with the address of the first byte of the data block (acting as a pointer). Using the start address as a pointer, multiple data blocks can be used, and any area of RAM may be used. Given this, the RAM data structure is re-locatable and its location will be defined by the user. The control and data bytes are described in [Table 4](#). Its important to note that variable naming will also vary compared to instance A.

Table 4. Variables Used in ROM-Resident Routines (Instance B)

Variable Name	Description	Location
FILE_PTR	Registers H and X are initialized with the address to the first byte of the data block in RAM.	H:X
BUS_SPD	Indicates the operating bus speed of the MCU. The value on this byte should equal to 4 times the bus speed.	FILE_PTR
DATASIZE	This byte indicates the number of bytes in the data array to be manipulated. The maximum data array size is 255 bytes.	FILE_PTR + 1
ADDR	These two bytes indicate the start address of the FLASH memory to be manipulated (ADDRH:ADDRL).	FILE_PTR + 2
DATA	This data array contains data that is to be manipulated.	FILE_PTR + 4

ROM-Resident Routines

For instance B, there are only three routines that may be used. PRGRNGE and ERARNGE are used in a similar way as in instance A. [Table 5](#) shows a summary of the ROM-resident routines.

Table 5. Summary of ROM-Resident Routines

Routine Name	Description
PRGRNGE	Program a range of FLASH locations with data loaded into the DATA array ⁽¹⁾ .
ERARNGE	Erase a page or a whole array of FLASH.
LDRNGE	Load the data array in RAM with data from a range of FLASH locations.

1. FLASH locations must be blank so they can be written to. User must ensure that the range specified is first erased before attempting to program it.

Addresses of Routines

Table 6 provides necessary addresses used in the on-chip FLASH routines for the HC908 families that use instance B.

Table 6. MCU Type vs. On-Chip FLASH Routines Addresses (Instance B)

MCU	ROM-Resident Routines			
	PRGRNGE	ERARNGE	LDRGNE	DELNUS
JK8/JL8	\$FC06	\$FCBE	\$FF30	\$FD35
LJ12	\$FC06	\$FCBE	\$FF30	\$FD35
LJ24/LK24	\$FC06	\$FCBE	\$FF30	\$FD35
AP	\$FC34	\$FCE4	\$FC00	\$FF2C
JW32	\$FE10	\$FE13	\$FA31	—

Note: Refer to AN2272 for a more detailed description on using these routines.

Code Example

The following C code is a basic example using a 68HC908QY4 (instance A implementation of the ROM-resident routines). This piece of code configures and uses the PRGRNGE routine to program 4 bytes into a specific location in FLASH. Code for instance B implementation is not described in this section, but its usage is very similar in the provided implementation.

First, the address for the PRGRNGE routine is defined at address \$2809 (as shown in Table 3 for QY4):

```
/* ROM Resident Routines Pointer Definitions */
#define PRGRNGE    0x2809    /* Program a range of FLASH locations */
```

Next, the RAM data block must be defined and placed at address \$88. FADDR is also defined and initialized with the first address in FLASH to be programmed. This variable will be used as a parameter when calling PRGRNGE:

```
/* RAM Data Block */
unsigned char CTRLBYT @0x0088;    /* Select between mass/page erase */
unsigned char CPUSPD @0x0089;    /* CPU value equals 4x Bus speed */
unsigned int  LADDR @0x008A;    /* Last address of FLASH to manipulate */
unsigned char DATA[4] @0x008C;  /* Data to program into FLASH */

unsigned int FADDR = 0xEF00;    /* First Address to store data in FLASH */
unsigned char DataSize = 4;    /* Variable to calculate Last Address */
```

As soon as the RAM data block is defined, it must be initialized:

1. CTRLBYT is set to page erase (which is not used in this exercise, because there is no erasure of FLASH locations, only data programming).
2. CPUSPD is calculated (f_{op} times 4); based on an internal oscillator frequency of 3.2 MHz, the result is 12.8, rounded up to 13.
3. LADDR will be the last address in FLASH to be programmed. Because we want to write 4 data bytes, calculate this value adding 4 to FADDR.

```

/* RAM DATA Initialization */
CTRLBYT = 0;          /* Page erase (clear bit-6) */
CPUSPD = 13;         /* 3.2MHz * 4 = 12.8 = 13 (using internal OSC) */
LADDR = FADDR + DataSize - 1; /* Last address in FLASH to program */

```

The programming routine FLASHProgram will load in H:X the value from FADDR, and then jump to the PRGRNGE subroutine to start executing from ROM using the data in the RAM data block as parameters.

```

void FLASHProgram(void){

    asm(LDHX FADDR);    /* Load address of RAM Data Block to H:X */
    asm(JSR PRGRNGE);  /* Call PRGRNGE ROM Subroutine */

}

```

Considerations

AN2874SW.zip, the example code provided with this application note, was developed using CodeWarrior IDE version 5.0 for HC08. There may be small changes needed for the code to be used in a different MCU. Depending on the HC908 being used, the addresses will vary, as previously stated in this document. Two different projects are included: one created for the MC68HC908QY4 (instance A), and one created for the MC68HC908LJ12 (instance B).

This document describes how to use the ROM-resident routines to program data into the FLASH using different families of HC908s. It is important to highlight the fact that some families in the HC908 will use one implementation or the other, and that not all 68HC908 will have these routines embedded in ROM. On-chip FLASH routines are not supported in the following families: AB, AS/AZ, BD, LD, MR, GP, SR, GZ32/48/60, and GR32/48/60.

It's also important to note that some MCUs will require the flash block protection register (FLBPR) to be configured before attempting a write/erase to FLASH. In some cases, this register will be set to protect the entire FLASH memory by default. Please refer to the FLASH section of the data sheet for the particular MCU family to check the out-of-reset state of this register.

References

Visit www.freescale.com for the most up-to-date versions of these and other useful documents:

AN2874SW.zip — Contains example code for the routines described in this application note
MC68HC908LJ12/D, Technical Data Sheet — Section 10.6 ROM-Resident Routines
AN1831 — Using MC68HC908 On-Chip FLASH Programming Routines.
AN2272 — In-Circuit Programming of FLASH Memory in the MC68HC908LJ12
AN2295 — Developer's Serial Bootloader for M68HC08 and HCS08 MCUs
AN2346 — EEPROM Emulation Using FLASH in MC68HC908QY/QT MCUs
AN2504 — On-Chip FLASH Programming API for CodeWarrior Software
AN2545 — Using MC68HC908GR/GZ On-Chip FLASH Programming Routines
AN2635 — On-Chip FLASH Programming Routines for MC68HC908LB8, MC68HC908QL4,
MC68HC908QB4, MC68HC908QB8, MC68HC908QT5, and MC68HC908QY8
EB618 — Typical Data Retention for Nonvolatile Memory
EB619 — Typical Endurance for Nonvolatile Memory

How to Reach Us:

Home Page:

www.freescale.com

E-mail:

support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:

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

Japan:

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

Asia/Pacific:

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

For Literature Requests Only:

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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor 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 that may be provided in Freescale Semiconductor 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. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor 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 Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor 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 Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2004. All rights reserved.