Order this document
by AN1831/D

Motorola Semiconductor Application Note

AN1831

Using MC68HC908 On-Chip FLASH Programming Routines

ROM-Resident Routines in the MC68HC908GR8, MC68HC908KX8, MC68HC908JL3, MC68HC908JK3, and the MC68HC908JB8

By Grant Whitacre
MMD Applications Engineering
Austin, Texas

Introduction

This application note describes how to use the routines that are stored in ROM (read-only memory) in the MC68HC908GR8, MC68HC908KX8, MC68HC908JL3/JK3, and the MC68HC908JB8 microcontrollers (MCU).

These routines are used to program, erase, and verify FLASH memory and may be accessed in either user mode or monitor mode[1]. There are additional routines in the MC68HC908KX8 to trim the internal clock generator, which are also described herein. This document describes the method of calling each of the routines in the collection and specifies what is performed and returned as confirmation of routine execution.

To illustrate how these routines are used in practice, a program is included, which can be configured for use in any of these devices to program FLASH in either user mode or monitor mode.

1. These routines are accessible in both user mode and monitor mode in all listed devices except the MC68HC908GR8. This device allows access to these routines in monitor mode only.

In addition, a host program, downloadable from the Motorola Web site, has been developed to provide a PC interface to download this program to a device to program FLASH.

FLASH Overview

The routines described here have been incorporated into ROM on these particular devices, which do not have enough RAM to allow for this functionality in a RAM routine. The type of FLASH for which these routines are applicable is called "split gate" FLASH because of the type of technology used, TSMC FLASH after the fabrication plant, or SST FLASH after the company who originally designed it.

Split gate FLASH has significant advantages. Some of these advantages are:

- Faster programming time. It takes 30 to 40 μs to program each byte, which translates to a little more than a quarter second of programming time to program an entire 8-Kbyte array.
- Better endurance. This type of FLASH is specified to withstand at least 10,000 program/erase cycles. Older technologies provided only about 100 program/erase cycles.
- Simpler programming algorithm. The programming algorithm for split gate FLASH is a simple process of turning on high voltage, applying it to the row to be programmed, and writing values to each byte to be programmed in turn. This differs from past technology which required an iterative process of turning on high voltage and applying it to a page, writing values to each byte in the page, checking all bytes for valid values in a "margin" read condition, and then repeating the program/verify process until all bytes are verified correctly.

Split gate FLASH is programmed generally on a row basis and erased on a page basis. Also, the entire array can be mass erased. A page always contains two rows, but the size of the page can vary from one device to another. A typical page size is 64 or 128 bytes. Before reprogramming a byte in one row that is currently programmed with a

different value, the entire page must be erased and reprogrammed. Refer to the applicable data manual for the proper program and erase procedure for this FLASH.

The Routines

The collection consists of five callable[2] routines and each is described in **Table 1**. These routines are explained briefly here, but the parameters and the passing method are addressed in later sections.

GETBYTE

GETBYTE is a routine that receives a byte on the monitor mode communication port defined for that particular device, and this received value is passed back to the calling routine in the accumulator. For these devices, the communication port is either port A0 or port B0. Check **Table 3** for the constant definition for COMMPORT for the port used for each device. This routine expects the same non-return-to-zero (NRZ) communication protocol and baud rate that is used in monitor mode[3]. The difference between this routine's method of receiving a byte and when the monitor receives a byte is that the monitor echoes back whatever is received. It may be more efficient for a RAM program to use this routine when receiving data from a host, to eliminate the time overhead in sending out every byte that is received. This is especially true if the host program and RAM routine already have a built-in error detection scheme, such as a message checksum, and there might not be a need to do an echo check for each byte sent.

2. These routines are accessible in both user mode and monitor mode in all listed devices except the MC68HC908GR8. This device allows access to these routines in monitor mode only.

3. The baud rate will be $f_{OP}/256$ for all but the MC68HC908JB8. In this device, the bit rate for this routine as well as for the monitor mode send/receive routines have been changed to accommodate a "standard" f_{OP} for this device considering it is a USB part. The bit rate for the MC68HC908JB8 is $f_{OP}/208$.

RDVRRNG

RDVRRNG routine serves two purposes:

- It can be used to read a range of FLASH locations.
- It can be used to verify a range of FLASH locations with data contained in the data array in RAM.

Actually, both functions are performed each time the routine is called, and the data in the specified FLASH range is returned. A degree of flexibility with this routine is that one can specify where the data is to be returned. If the accumulator is 0 when entering RDVRRNG, then the data read will be sent to the monitor mode communication port. If the accumulator is non-zero, then the data is placed in RAM in the data array, replacing the existing contents. The beginning and end of the range to be read and/or verified are specified as parameters to this routine. The carry bit of the condition code register is set if the data in the specified range is verified successfully against the data in the data array. One more added function of this routine is that it does a checksum on the data returned. This checksum, which is the LSB of the sum of all bytes in the entire data collection, is stored in the accumulator upon return from the function.

PRGRNGE

PRGRNGE is used to program a range of FLASH locations with data loaded into the data array. As with RDVRRNG, the start and end location of the range of addresses to be programmed is passed by parameter. A check to see that all bytes in the specified range are erased is not performed by this routine prior to programming. Nor does this routine do a verification after programming, so there is no return confirmation that programming was successful. It should be noted that PRGRNGE returns with the first-address variable, FADDR, set to the address of the next byte after the range just programmed. The last-address variable LADDR is not changed. Also, since this routine calls the delay routine DELNUS, parameter passing requirements for that routine must be met when calling PRGRNGE.

Another point worth noting is that this routine allows any range to be passed to it. That is, the range does not have to be coincident with row boundaries. The range specified can be at the beginning of a row, the middle of a row, the end of a row, or it can be a range overlapping row

boundaries. The only two things that the user must assure is that the range specified is first erased and that whatever is specified as the range, the data for the range must be in the data array in RAM.

NOTE: *This routine can be used in conjunction with RDVRRNG to perform a complete program and verification cycle of the specified range.*

ERARNGE

ERARNGE can be called to erase a range of locations in FLASH. This routine does not use the last address (LADDR) variable. The first address (FADDR) placed in H:X in the two previous routines actually can be any address in the range to be erased. There are only two sizes of erase ranges: a page or the entire array. Therefore, this routine needs to be told what type of erase is desired. This is done by setting a bit called the mass bit in a control variable in RAM called CTRLBYT. This is explained in more detail later in **Variables**.

DELNUS

The last routine is a delay routine used in support of the PRGRNGE and ERARNGE routines. It can, however, be called independently. DELNUS takes two parameters – one signifying the operating frequency passed via the accumulator and the other, a single byte value passed in the X register, specifying the length of the delay. Neither of these parameters is passed as an absolute value. The operating frequency variable is a value four times that of f_{OP} actually used, and this value has an allowable lower limit of four representing 1-MHz operation. The delay value passed represents the number of 12 microsecond increments for the delay. Therefore, the resolution of the delay is 12 microseconds. The minimum delay is, of course, 12 microseconds and the maximum delay for this routine is a little more than 3 milliseconds (255 * 12 μs). The precision of the delay is very high considering that it is normalized to the frequency of operation which can be specified to within 0.25 MHz. The worst precision occurs for short delays at relatively slow operating frequencies, where both values passed are midway between possible values[4].

4. An example of this worst-case error would be an f_{OP} of 1.125 MHz and a desired delay of 18 μs. For these conditions, a value for the frequency parameter could be either 4 or 5, signifying an f_{OP} of 1.00 or 1.25 MHz, respectively. The delay value passed could be either 1 or 2, signifying 12 or 24 μs delay, respectively. In a case like this, choose the lower value for one parameter and the upper value for the other parameter to minimize the error of the delay.

When the delay routine is called by PRGRNGE or ERARNGE (the only routines in this collection which call the delay routine), the calling routine loads the X register with the value of the delay needed. The frequency parameter is loaded into the accumulator by reading the RAM variable CPUSPD. This variable, therefore, must be pre-loaded by the RAM routine calling PRGRNGE or ERARNGE.

Table 1. FLASH Routines

Routine Name	GETBYTE	RDVRRNG	PRGRNGE	ERARNGE	DELNUS
Routine Description	Gets a byte of data from comm port	Reads and/or verifies a range of locations	Programs a range* of locations	Erases** a page or the entire range	Delays for n x 12 µs
Entry Conditions	Comm port configured as an input	H:X contains first address of range; LADDR contains last address read; Acc is tested to see if read data goes to comm port (Acc = $00) or to DATA;DATA contains data against which to compare read data	H-X contains first address of range; LADDR contains last address to be programmed; DATA contains data used during programming; CPUSPD contains 4 * f_{OP}	H:X contains any address in range to be erased; range size specified by control byte; CPUSPD contains 4 * f_{OP}	X contains time ÷ 12 of delay (in µs); Acc contains 4 times f_{OP}
Exit Conditions	Acc is loaded with byte received	C bit is set if good compare; Acc contains checksum; DATA may contain read FLASH data	H:X contains next address after range just programmed	Preserves contents of H:X (address passed)	
Subroutines Called	Get_Bit		DELNUS	DELNUS	
Variables Read		LADDR, DATA ARRAY	CONTROL BYTE, LADDR, DATA ARRAY, CPUSPD	Control Byte, CPUSPD	
Variables Modified		DATA ARRAY			
Stack Used	4 bytes	6 bytes	7 bytes	5 bytes	3 bytes

*Allows programming of a range of addresses, which does not have to be on a row boundary, either beginning or end. For example, programming $F001 to $F008 is valid.

** Does not check for a blank range before (to see if erase is necessary) or after (to see if successful erase)

Defined Constants

Table 2 lists the various constants defined for these routines. All but the FLCR address relate to delays used during programming and erasing. The constants ending in a Q are values passed to the delay routine. As mentioned previously, the delay routine takes a parameter which represents the number of 12 microsecond increments of delay time. Therefore, program time, TPROG, which is specified as a time between 30 and 40 microseconds, has a duration here of 12 times TPROGQ, or 36 microseconds.

Page erase and mass erase delays are done the same way, except that the routines are called ECALLS and MECALLS times, respectively. Therefore, a mass erase delay, which is specified to be 4000 microseconds, is actually 20 delays each with a duration of 17 * 12 microseconds, which results in a total mass erase delay of 4080 microseconds (MECALLS * TMERASEQ * 12 microseconds).

Table 2. Constants Used in Routines

Constant Name	Description	Value
FLCR	FLASH control register address	$FE08
TPROGQ	Program time	3
TERASEQ	Erase time	17
TMERASEQ	Mass erase time	17
TNVSQ	HVEN setup time	1
TPGSQ	Program hold time	1
TNVHQ	HV hold time	1
TNVHLQ	HV hold time (mass erase)	8
TRCVQ	Return to read time	1
ECALLS	Calls to delay for page erase	5
MECALLS	Calls to delay for mass erase	20

Because of the differences in some of the constants used for each device, the following constants need to be specific to the particular device. **Table 3** shows the constant values for each device. Since these values are device-specific, they have not been included in the source code in **ROM Routines Source Code**.

Table 3. Device-Specific Values for Constants

Constant Name	Description	MC68HC 908GR8	MC68HC 908KX8	MC68HC 908JL3/JK3	MC68HC 908JB8
RAM	Start address of RAM	$40	$40	$80	$40
ROWSIZ	Size of row for programming	32	32	32	64
COMMPORT	Communication port for monitor mode	PTA0	PTA0	PTB0	PTA0
FLBPR	FLASH block protect register address	$FF7E	$FF7E	$FE09	$FE09
Get_Put	Address of routine to get and then output a byte on the comm port (monitor code)	$FE99	$FE97	$FEBD	$FEC0
Put_Byte	Address of routine to output a byte on communication port (monitor code)	$FEAE	$FEAA	$FED0	$FED5
Get_Bit	Address of routine to get a bit on communication port (monitor code)	$FED2	$FECE	$FF00	$FF00

Variables

Table 4 shows the variables used in the routines. These variables are either passed in a register or as static variables in a predefined location in RAM. FADDR is a 2-byte value that represents the first address in the range on which to be operated. It is passed in the H:X registers when a call is made to one of the routines. The first address of a range can be any valid FLASH address and does not have to be on a row or page boundary.

LADDR is the last address in the range and is passed in the first byte of the data structure in RAM. This data structure is very simple, consisting of the last address, the CPU speed variable, a control byte, and the data array. It is discussed in detail in **The Data Structure**. The last address, like the first address, can be any valid FLASH address and is not restricted to being the last byte of a page or row.

The internal operating frequency of the device on which the FLASH operation is to be performed is passed in a variable called CPUSPD. It is a 1-byte value which is passed in the data structure and should be given as the rounded product of four times the actual internal operating frequency, such that if f_{OP} is 2.4576 MHz, then the value passed should be decimal 10, or $0A. This variable is used to normalize the length of delays with respect to the operating frequency, and passing a value four times the actual frequency provides better resolution.

The remaining operating parameter used in these routines is a single bit value in the control byte. This bit is called the mass bit and is set when calling ERARNGE to perform a mass erase. If ERARNGE is called with the intention of performing a page erase, then the mass bit must be cleared. The other bits in CTRLBYT are not used and can be set at the user's discretion for other flags.

Table 4. Variables Used in Routines

Variable Name	Description	Size	Location/Passing Method
FADDR	First address of range of locations	2 bytes	H:X
LADDR	Last address of range of locations	2 bytes	Data structure
CPUSPD	4 x f_{OP}	1 byte	Data structure
CTRLBYT	Mass bit (bit 6)	1 byte	Data structure
DATA	Data array	Variable	Data structure

The Data Structure

The data structure is a collection of static variables in RAM used in the execution of the three main routines – PRGRNGE, ERARNGE, and RDVRRNGE. The data structure is in the same relative location in RAM and the content is the same data and order for all of the devices containing these ROM routines. The structure always starts in the ninth byte of RAM and the order of the variables is as shown in **Table 5**.

Table 5. Data Structure Location and Content

Location	Variable Name	Size (Bytes)	Description
RAM + $08	CTRLBYT	1	Includes mass flag as bit 6
RAM + $09	CPUSPD	1	CPU speed passed as 4 x f_{OP}
RAM + $0A RAM + $0B	LADDR	2	Last address for read a range and program a range
RAM + $0C	DATA	Variable	Variable number of bytes of passed data for programming or verifying a block

Note that the data array DATA is variable in length. This is done to support a variable number of locations on which to perform any of the programming, reading, or verifying actions. Most of the time, these actions will be performed on a row of data at one time, although that need not be the case. Some of these devices have a rather small RAM array, and the size of the data array must be limited to the size of RAM minus the stack needed and the size of any RAM routine being executed. If the RAM routine is kept to a reasonable size, then there should not be a problem defining the data array to be the size of a row for any of the devices in this collection.

Addresses of Routines

The address to call each of the five routines varies among the devices. **Table 6** gives the absolute address that should be used when calling the routines.

Table 6. Addresses of Routines

Routine	MC6868HC 908GR8	MC68HC 908KX8	MC68HC 908JL3/JK3	MC68HC 908JB8
GETBYTE	$1C00	$1000	$FC00	$FC00
RDVRRNG	$1C03	$1003	$FC03	$FC03
ERARNGE	$1C06	$1006	$FC06	$FC06
PRGRNGE	$1C09	$1009	$FC09	$FC09
DELNUS	$1C0C	$100C	$FC0C	$FC0C

MC68HC908KX8 Trim Routine

The MC68HC908KX8 contains two additional routines in ROM, which have been included to support the trimming of the internal clock generator (ICG) module. ICGTRIM is located at $1330 in the MC68HC908KX8 and can be called to trim the ICG by measuring the pulse width of a break signal received on port A0 or port B4. The baud rate used for the break signal must be equal to the internal frequency of the device divided by 256. Communication must be in conformance with normal monitor mode communication, that is, non-return-to-zero (NRZ) format. A break signal is defined as 10 consecutive low bits, so the pulse width of this signal is nominally 1.04 milliseconds at 9600 baud. This signal must be within 25 percent of the nominal value or the routine will not attempt to trim the ICG.

Table 7 specifies the relationship between the internal frequency, the baud rate, and the pulse width of the break signal.

Table 7. Frequency, Baud Rate, Break Pulse Width

f_{OP} (MHz)	Baud Rate (bps)	Break Pulse Width (ms)		
		Minimum	Nominal	Maximum
1.2288	4800	1.5623	2.083	2.604
2.4576	9600	0.781	1.042	1.302
3.6864	14400	0.365	0.521	0.651
4.9152	19200	0.195	0.260	0.325
7.3728	28800	0.098	0.130	0.163

This routine checks to see how many cycles are measured during a break signal (10 low bits) sent at $f_{OP}/256$ baud by a host and adjusts its trim register. If the break signal is more than 25 percent variation from what is expected (0.78-1.30 ms @ 9600), then ICG trimming will not be performed. This ICG accuracy limit is consistent with the extent of the ICG's ability to fine tune the trim register.

The main timing loop of this routine begins at the leading edge of the break signal and lasts until it sees the trailing edge. The break signal lasts for 10 bit times. Since communicating at $f_{OP} \div 256$ bps, then the duration of 10 bit times is 2560 cycles. Each time through the loop is 10 cycles, so it is expected to execute the loop 256 times if the MC68HC908KX8 is in sync serially with the host.

If the loop is executed for more than 256 loop cycles, then the MC68HC908KX8 must be running faster than expected and needs to be slowed down. If the loop is executed for less than 256 loop cycles, then the MC68HC908KX8 must be running slower than expected and needs to be speeded up. The amount that the CPU speed is changed is equal to the number of loop cycles over or under 256. So if the loop is traversed 240 times, then we are running $(256 – 240) \div 256 = 6.25$ percent fast.

Each incremental change that is made to the trim register (ICGTR) will result in a 0.195 percent change to the internal clock. That is, incrementing the register by one over the default value of $80 stored there will decrease the internal clock by 0.195 percent. Each execution of the loop over or under what is expected (256 times) represents an error of 1/256 = 0.391 percent error. So the number of loop cycles is

doubled and this number is used to correct the trim register. The precision for trimming is therefore 0.391 percent.

Another routine that is unique to the MC68HC908KX8 is called ICGTEST. This routine simply toggles a port pin, port A4, at a rate that is 1/16th of the operating frequency. This allows verification that the ICG was trimmed accurately. ICGTEST is located at $1369.

Typical Routine Calls

This section provides examples of how these routines may be called.

The following code makes a call to the delay routine (DELNUS). Assume f_{OP} = 7.37 MHz, so the value passed in the accumulator is round (f_{OP} * 4) = 29 ($1D). The delay value is loaded into and passed through the X register. For example, let's use a value, TMERASEQ, which is the desired delay time divided by 12.

```
DELAYCALL:
       LDA    #$1D                    ;fOP*4
       LDX    #TMERASEQ               ;delay time/12
       JSR    DELNUS
```

The next block of code makes a call to the routine RDVRRNG to read and verify a range of FLASH from $F000 to $F010. The accumulator is cleared before calling the routine, which signals to the routine that the specified range is to be sent out the communication port instead of being copied into RAM.

The verify stage will be performed automatically and each byte in the FLASH range will be compared to the corresponding byte in the data array in RAM. That is, the first byte of the range, $F000, will be compared with the first byte in the data array which is located at the 13th byte of RAM by definition. This process is repeated for all bytes in the range and if any of the comparisons is not equal, then the carry bit of the condition code register will be cleared upon return from RDVRRNG. Otherwise, it will be set. This code does not show the loading of the compare data into RAM.

Before calling the routine, the high byte and low byte of the last address of the range are placed in the 11th and 12th locations of RAM, respectively, and the H:X register is loaded with the first address of the range.

```
RDCALL:
     CLRA                    ;COMMPORT IS DEST.
     LDHX   #$F010           ;LAST ADDRESS IS STORED AT LADDR
     STHX   LADDR
     LDHX   #$F000           ;FIRST ADDRESS IS STORED IN H:X
     JSR    RDVRRNG
```

The next few lines of code perform an erase of FLASH. The variable CPUSPD located at the 10th location of RAM is set to a value which reflects an 8-MHz operating frequency, that is 8 * 4 = 32 ($20). Since we are calling the erase routine, we must specify what type of erase we want to do: page erase or mass erase. This example illustrates the setup to perform a mass erase where the mass bit, bit 6, in CTRLBYT at the ninth location of RAM must be set. Any valid FLASH address is loaded into H:X when doing a mass erase. In the case of a page erase, any address within that page would be acceptable.

```
MASSERASE:
       MOV   #$20,CPUSPD    ;SET CLOCK VALUE AT 8 MHZ
       BSET6 CTRLBYT         ;SET TO MASS ERASE HERE
       LDHX  #$F000          ;LOAD ANY FLASH ADDRESS IN H:X
       JSR   ERARNGE
```

To call GETBYTE to receive a byte of data on the communication port, the only thing that needs to be done is to ensure that the communication port is configured as an input. The next code example assumes that port A0 is the communication port.

```
RECEIVEBYTE:
       BCLR   0, DDRA        ;CLEAR BIT 0 DATA DIRECTION
                             ; REGISTER FOR INPUT ON PTA0
       JSR    GETBYTE
```

The final two examples show how to call the ICG trim routine resident in MC68HC908KX8 ROM, and then call the test routine to verify the accuracy of the internal clock. To set up for the call to trim the ICG, several things must be done. First, we make sure that the ICG is enabled (ICGON bit in the ICG trim register is set) and the internal clock is selected (CS bit in the trim register is cleared). Then the accumulator is

set to select the port which is to receive the break signal. In this example, port A0 is used as the communication port and the one where the break signal will be received. To select port A0, the accumulator must contain a non-zero value. We'll also set this port as an input here.

```
TRIMTHEICG:
       BCLR    0,DDRA          ;SET PTA0 AS AN INPUT
       MOV     #$80,ICGTR      ;SET THE TRIM REGISTER TO MIDPOINT
       MOV     #$08,ICGCR      ;TURN ON THE ICG AND SELECT IT A
                               ; CLOCK SOURCE
       LDA    #$FF             ;ANY NON-ZERO VALUE TO SELECT PTA0
                               ; FOR COMM
       JSR    ICGTRIM
```

There is no setup required to call the next routine which allows monitoring of a set fraction of the operating frequency. The port used to output 1/16th the operating frequency, port A4, is set as an output in the routine. Therefore, only the call is required. To stop execution of this routine, the IRQ pin needs to be pulled low. External interrupts can be disabled (I bit set in the CCR) so as not to generate an inadvertent interrupt when this pin is set low to exit this routine.

```
TESTTHEICG:
       JSR    ICGTEST
```

Example RAM Routine

This section describes a program containing a RAM routine which could be used in either monitor mode or user mode for the purpose of programming one of these devices. In monitor mode, the routine could be downloaded via monitor commands and in user mode the routine could be copied to RAM from FLASH.

Those readers who have read *In-Circuit Programming of FLASH Memory in the MC68HC908GP20*, Motorola document order number AN1770/D, will recognize the content and structure of this program. Refer to AN1770 for a complete description of the protocol used to send programming commands and data to this routine. The PC-based host program described in that application note has been expanded to support the programming of these and other devices and is available in

the software library of the Motorola Web site at: http://mot-sps.com/csic/techhelp/appsw/appsw.htm.

The RAM routine here is much smaller than that required for the MC68HC908GP20 because it makes calls to the ROM routine rather than have these routines included in the RAM routine. The latter situation would not be practical in small RAM-array devices such as the ones that include these routines. The source code for this program follows. The user of this routine must make sure that the assembler directives are set properly based on the device and the mode to be used.

This routine also differs from the GPZO's in that it only supports monitor comm port communication for both user and monitor mode programming. Since the SCI is not available on two of these devices, SCI communication is not described here. This program could be modified easily to support user mode SCI programming.

This program does not include support for trimming the ICG in the MC68HC908KX8. A RAM routine for monitor mode trimming or a FLASH-based routine for user mode trimming could be generated by the user. Note though that the host program referred to previously can be used to send the break signal for automatic trimming.

```
*****************************************************************************
* FILE NAME: GKJJRR.ASM
* PURPOSE: Provides a FLASH erase, program, and verify program
* TARGET DEVICE: MC68HC908GR8, MC68HC908KX8, MC68HC908JL3/JK3 and the MC68HC908JB8
*
* ASSEMBLER: mcuEZ
* VERSION: 1.0.5
*
* PROGRAM DESCRIPTION:
* This program loads a RAM routine with instructions/data
* located in FLASH memory that:
*       Receives data over the monitor comm. Port
*       Calls ROM routine to program FLASH with received data
*       Calls ROM routine to read/verify a FLASH range
*       Calls ROM routine to bulk erase device upon command
*
* The program has assembler directives to be able to program each device in both
* user and monitor modes. In monitor mode, the generated S-record file will contain
* only the RAM routine. It will not have any code that would reside out of RAM.
* In user mode, load routines are incorporated so that it could be contained in a
* user's application. The load routines load the programming routines into RAM and
* from there it looks just like the RAM routine executed in monitor mode.
```

```
*
*
* AUTHOR: Grant Whitacre
* LOCATION: Austin, Texas
*
* UPDATE HISTORY:
* REV      AUTHOR         DATE          DESCRIPTION OF CHANGE
* ===   ============    ========       ====================
* 0.0   G. WHITACRE     11/02/98       INITIAL VERSION
* 0.1   G. WHITACRE     01/19/99       MOD. FOR KX6
* 0.2   G. WHITACRE     04/22/99       MOD. FOR JL3
* 0.3   G. WHITACRE     11/18/99       MOD. FOR JB8, GR8
*
* GENERAL CODING NOTES:
* Bit names are labeled with <port name><bit number> and are used in the commands
* that operate on individual bits, such as BSET and BCLR. A bit name followed by a
* dot indicates a label that will be used to form a bit mask.
*
*******************************************************************************
* ASSEMBLER DIRECTIVES
* (INCLUDES, BASE, MACROS, SETS, CONDITIONS, RAM DEFS, ETC.)
*******************************************************************************
 BASE 10D                                    ;DEFAULT TO BASE 10 NUMBER DESIGNATION

                                             ;Remember: ACTIVE LOW!!!!!!!!!!!!!!!
RAMPROG: SET 0                               ;IF SET, ALL (NECESSARY) ROUTINES WILL BE
                                             ;ADDRESSED IN RAM INITIALLY;THIS VERSION
                                             ;WOULD BE USED AS THE S19 RECORD FILE
                                             ;THAT IS DOWNLOADED INTO RAM IN MONITOR
                                             ;MODE FOR FLASH PROGRAMMING

* SELECT ONLY ONE OF THE FOLLOWING!
GR8: SET 1                                   ;SELECTS GR8 AS THE TARGET DEVICE
KX8: SET 1                                   ;SELECTS KX8 AS THE TARGET DEVICE
JB8: SET 1                                   ;SELECTS JB8 AS THE TARGET DEVICE
JL3: SET 0                                   ;SELECTS JL3 AS THE TARGET DEVICE
*******************************************************************************
* APPLICATION-SPECIFIC MEMORY AND I/O EQUATES
*******************************************************************************
* THE VALUE FOR SPDSET, WHICH IS THE fOP*4, normalizes delay routines
* to an absolute time.
SPDSET          EQU     10                   ;10 => 2.5 MHZ OPER. FREQ.
PTA             EQU     $00
PTB             EQU     $01
CONFIG1         EQU     $1F
MASSBIT         EQU     6                    ;CTRLBYT MASS BIT = 6
RAMPRSZ         EQU     $50                  ;NOT TO EXCEED SIZE OF RAM ROUTINE
RAMPRG          EQU     $AC                  ;START OF RAM ROUTINE
PRGSTRT         EQU     $F000                ;START OF FLASH PROGRAM
XFRCODE         EQU     PRGSTRT+RAMPRG
RSTVLOC         EQU     $FFFE                ;RESET VECTOR LOCATION
```

```
FLCR            EQU     $FE08               ;FLASH CONTROL REGISTER

       IFEQ GR8
COMPORT         EQU     PTA
RAM             EQU     $40
GETBYTE         EQU     $1C00
GET_PUT         EQU     $FE99
GET_BIT         EQU     $FED2
PUT_BYTE        EQU     $FEAE
ROWSIZ          EQU     32
FLBPR           EQU     $FF7E
       ENDIF
       IFEQ KX8
COMPORT         EQU    PTA
RAM             EQU    $40
GETBYTE         EQU    $1000
GET_PUT         EQU    $FE97
GET_BIT         EQU    $FECE
PUT_BYTE        EQU    $FEAA
ROWSIZ          EQU    32
FLBPR           EQU    $FF7E
       ENDIF
       IFEQ JL3
COMPORT         EQU    PTB
RAM             EQU    $80
GETBYTE         EQU    $FC00
GET_PUT         EQU    $FEBD
GET_BIT         EQU    $FF00
PUT_BYTE        EQU    $FED0
ROWSIZ          EQU    32
FLBPR           EQU    $FE09
       ENDIF
       IFEQ JB8
COMPORT         EQU    PTA
RAM             EQU    $40
GETBYTE         EQU    $FC00
GET_PUT         EQU    $FEC0
GET_BIT         EQU    $FF00
PUT_BYTE        EQU    $FED5
ROWSIZ          EQU    64
FLBPR           EQU    $FE09
       ENDIF

RDVRRNG         EQU    GETBYTE+3
ERARNGE         EQU    GETBYTE+6
PRGRNGE         EQU    GETBYTE+9
DELNUS          EQU    GETBYTE+12

DATSTRC         EQU    RAM+8          ;Leave 8-bit offset from start of RAM for dev tools
```

```
*******************************************************************************
* VARIABLE DEFINITIONS & RAM SPACE USAGE
*******************************************************************************
*                       DOWNLOADED      SET FOR RTNS            SIZE
*                       ----------      ------------------      ----------
* RAM - RAM+$07                         RES. FOR DEV. TOOLS     (  8 BYTES)
* RAM+$08               TRANSFER SIZE   CTRLBYT                 (  1 BYTE)
* RAM+$09               FIRST ADDRESS   CPUSPD                  (2/1 BYTE)
* RAM+$0A:RAM+$0B       DATA SIZE       LAST ADDRESS            (1/2 BYTES)
* RAM+$0C:RAM+$0D       DATA ARRAY      DATA ARRAY              ( 32 BYTES)
* $AC - $EB             RAM PROGRAM                             ( 64 BYTES)
* $EC-$FF               STACK                                   ( 20 BYTES)
*                                                       TOTAL   (128 BYTES)
                ORG     RAM
TEMP2B          RMB     2
TEMPH           RMB     1
TEMPL           RMB     1
                ORG     DATSTRC
CTRLBYT         RMB     1
CPUSPD          RMB     1
LADDR           RMB     2
DATA            RMB     ROWSIZ
*******************************************************************************
*     Program Algorithm (User Mode Programming)
*       1.      Initialize all variables and ports.
*       2.      Monitor COMM port for input of block of data to be programmed and
*               the start address. Load RAM with the data array (up to 64 bytes), the start
*              address and length of data array.
*       3.      Transfer the following subroutines to
*               RAM at address RAMPRG
*               A. LDDATA
*               B. MAINPRG
*       4.      Jump to first byte of main RAM program (RAMPRG).
*       5.      Execute RAM program MAINPRG and then return to comm
*               port monitoring loop in RAM.
*
*     Program Algorithm - Monitor Mode Programming
*       1.      Monitor comm port for input of block of data to be
*               programmed and the start address. Load RAM with the data array (up to
*               64 bytes), the start address and length of data array.
*       2.      Execute RAM program MAINPRG and then return to PTA0/PTB0
*               monitoring loop in RAM.
*******************************************************************************
* START OF PROGRAM
*******************************************************************************
   IFNE RAMPROG
        ORG     PRGSTRT
        CLR     COMPORT
        MOV     #$11,CONFIG1                ;DISABLE THE COP AND LVI
```

```
*******************************************************************************
* NAME: LDRAMPR
* PURPOSE: LOADS MAIN RAM PROGRAM AND ALL NEC. SUBROUTINES
* ENTRY CONDITIONS: NONE
* EXIT CONDITIONS: NONE
* SUBROUTINES CALLED:
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF FLASH
*******************************************************************************
LDRAMPR LDHX    #RAMPRG                     ;STORE THE START LOCATION IN RAM
        STHX    TEMPH                       ;WHERE CODE IS TO BE TRANSFERRED
        LDHX    #XFRCODE                    ;LOAD 1ST ADDR OF FLASH CODE TO BE
NXTMOVE MOV     X+,TEMP2B                   ;TRANSFER LOCATION IN RAM
        PSHH                                ;
        PSHX                                ;PUSH CURRENT FLASH ADDDR TO STACK
        LDHX     TEMPH                      ;LOAD ADDRESSES THAT HOLD THE DEST.
        MOV     TEMP2B,X+                   ;TRANSFER DATA FROM TRANSFER LOCATION
NEXT    STHX    TEMPH
        CPHX    #RAMPRG+RAMPRSZ             ;TO NEXT LOCATION AT RAM DESTINATION
        PULX                                ;POP CURRENT FLASH ADDR FROM STACK
        PULH
        BNE     NXTMOVE                     ;IF NOT DONE, CONTINUE
        JMP     RAMPRG

        ORG     XFRCODE                     ;START OF CODE TO BE TRANSFERRED TO RAM
   ELSE
       ORG     RAMPRG                       ;START OF MONITOR PROGRAM WHICH IS ORG'D
                                            ; IN RAM
   ENDIF
*******************************************************************************
* NAME: LDDATA
* PURPOSE: LOAD RAM WITH USER'S DATA AND START ADDRESS VIA THE COMM PORT;
*       PROGRAMS AND THEN DUMPS DATA THAT IS DOWNLOADED; ONLY DUMPS DATA
*       IN ROW SPECIFIED IF NUMBER OF BYTES TO BE PROGRAMMED (DATASIZ) IS 0.
* ENTRY CONDITIONS:
* EXIT CONDITIONS:
* SUBROUTINES CALLED: PRGFLSH, DUMPROW
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF RAM
* THE STRUCTURE OF THE DATA RECEIVED IS AS FOLLOWS:
*       LOCATION         DESCRIPTION                          RAM LOC.
*       ========        ==================================    ========
*       1               COUNT OF THE TOTAL NUMBER OF          RAM+$08
*                       BYTES TO BE SENT (INCL. THAT BYTE)
*       2-3             THE FIRST ADDRESS WHERE THE           RAM+$09 thru RAM+$0A
*                       FOLLOWING DATA IS TO BE PROGRAMMED
*       4               NUMBER OF BYTES TO BE PROGRAMMED      RAM+$0B
*       5-68            ARRAY SPACE FOR DATA TO BE PROGRAMMED RAM+$0C thru RAM+$4B
*
* IF A COUNT IS USED THAT IS GREATER THAN (PROGRAM LENGTH + 1)
* THEN THE ROUTINE WILL HANG AFTER THE LAST PROGRAM BYTE IS SENT.
```

```
* CONTINUOUSLY LOOPS LOOKING FOR NEW DATA ON THE COMM PORT. MUST RESET
* AFTER THE LAST ROW DOWNLOAD.
* IF A DATA ARRAY IS RECEIVED WITH A NUMBER OF BYTES TO BE PROGRAMMED OF $FF
* THEN PROGRAM WILL CONSTRUE THIS AS A SIGNAL TO ERASE THE ENTIRE ARRAY. THIS
* WAS THE MOST CONVENIENT WAY TO IMPLEMENT BULK ERASE WITHOUT HAVING TO HAVE
* A COMMAND BYTE IN THE DATA STRUCTURE.
*               TRANSFERRED             PROGRAM                 SIZE
*               ===================     ============            ===========
* RAM+$08       TRANSFER SIZE           CTRLBYT                 (  1 BYTE)
* RAM+$09       FIRST ADDRESS (MSB)     CPUSPD                  (  1 BYTE)
* RAM+$0A       FIRST ADDRESS (LSB)     LAST ADDRESS (MSB)      (  1 BYTE
* RAM+$0B       DATA SIZE (DATASIZ)     LAST ADDRESS (LSB)      (  1 BYTE)
* RAM+$0C-RAM+$4B  DATA ARRAY           DATA ARRAY              ( 64 BYTES)
*************************************************************************
LDDATA:
        CLRH
        LDX    #CTRLBYT                     ;POINT TO LOCATION OF TRANSFER SIZE
WAITRX: JSR    GET_PUT                      ;CALL TO ROUTINE IN MONITOR CODE
        CPX    #CTRLBYT                     ;BAD START - KEEP LOOPING FOR NON-0
        BNE    STORNOW
        TSTA
        BEQ    WAITRX
STORNOW STA    ,X                           ;STORE THE DATA IN RAM
        INCX                                ;MOVE TO NEXT RAM LOCATION
        DBNZ   CTRLBYT,WAITRX               ;DEC. PROG SIZE CNTR (1st BYTE)
                                            ;IF ENTIRE PROG NOT LODED, CONT.
CPARSE  LDHX   CPUSPD                       ;$89
        STHX   TEMP2B                       ;MAINTAIN FIRST BYTE IN TEMP2B
        MOV    #SPDSET,CPUSPD               ;PUT THE CPU SPEED SELECTED IN EQUATE
                                            ; INTO CPUSPD ADDR
        MOV    LADDR+1,TEMPH                ;MAINTAIN DATASIZ IN TEMP

        AIX    #ROWSIZ-1                    ;DO THIS FOR BOTH A DUMP OR A PROGRAM
        STHX   LADDR                        ;                 "
        LDHX   TEMP2B                       ;                 "

        LDA    TEMPH                        ;IF SIZE OF DATA TO BE PROGRAMMED
        BEQ    DUMPROW                      ;IS 0 THEN BRANCH TO DUMP
        COMA
        BEQ    ERASE1                       ;IF SIZE IS FFH, THEN BULK ERASE

JUSTPRG LDA    #$FF
        STA    FLBPR
        JSR    PRGRNGE
        BRA    DUMPROW

ERASE1  BSET   MASSBIT,CTRLBYT
        JSR    ERARNGE
```

```
DUMPROW LDHX   TEMP2B
        CLRA
        JSR    RDVRRNG
        BRA    LDDATA

   IFNE RAMPROG
*******************************************************************************
* INTERRUPT AND RESET VECTORS
*******************************************************************************
        ORG    RSTVLOC
RSTVEC  FDB    PRGSTRT
*******************************************************************************
   ENDIF
```

ROM Routines Source Code

The following five flowcharts provide graphic explanations of the ROM routines source code.

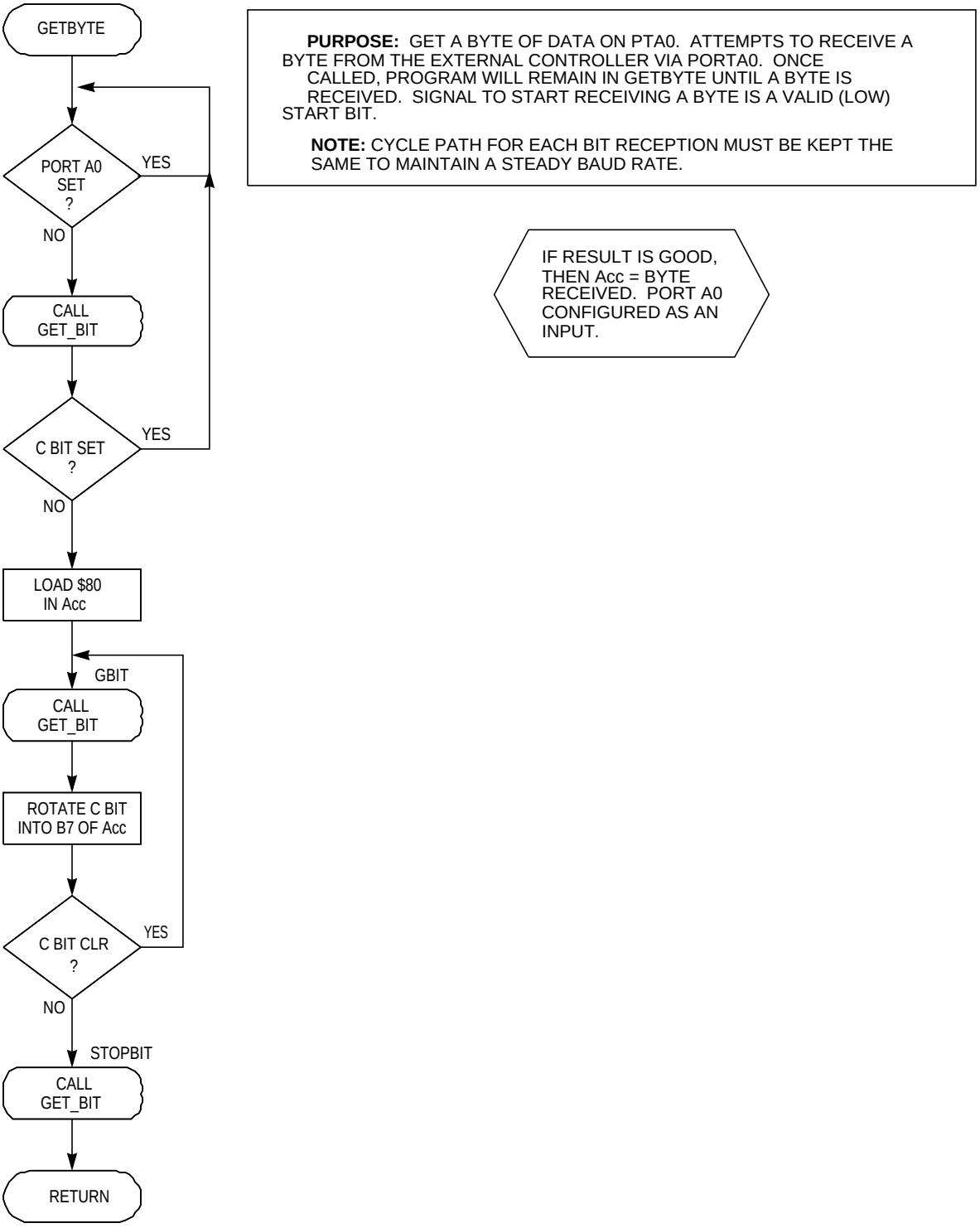


Figure 1. GETBYTE

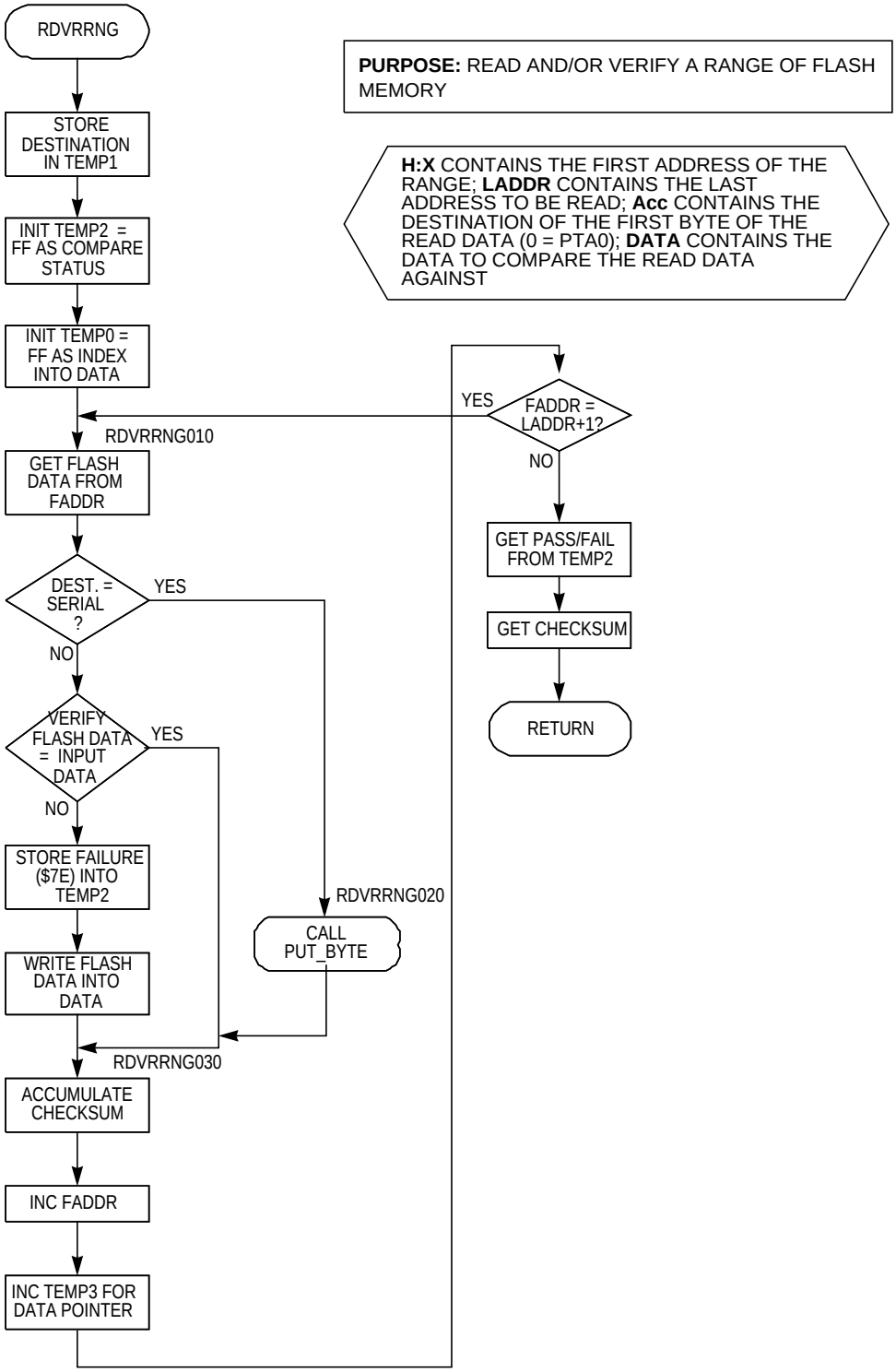


Figure 2. RDVRRNG

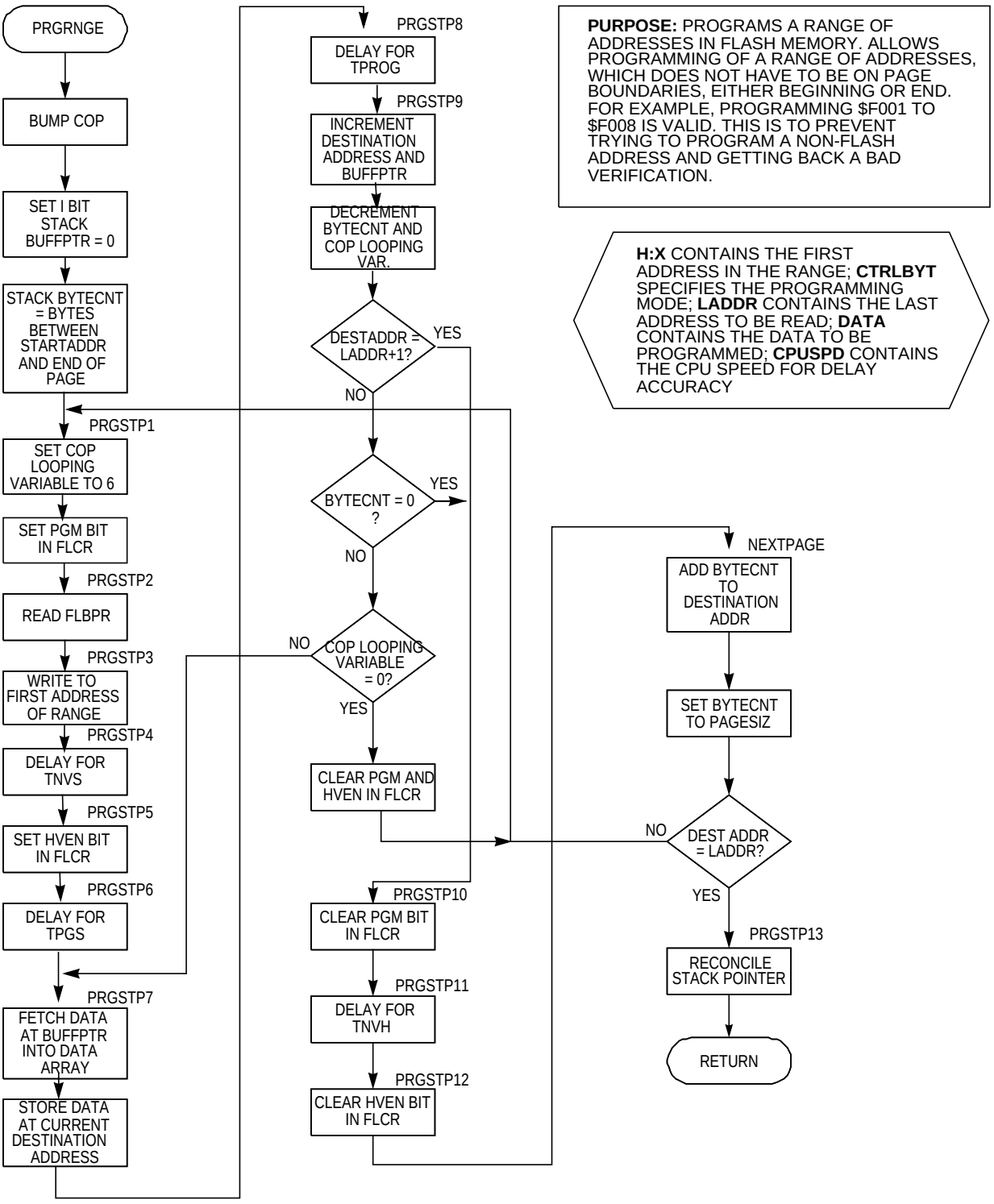
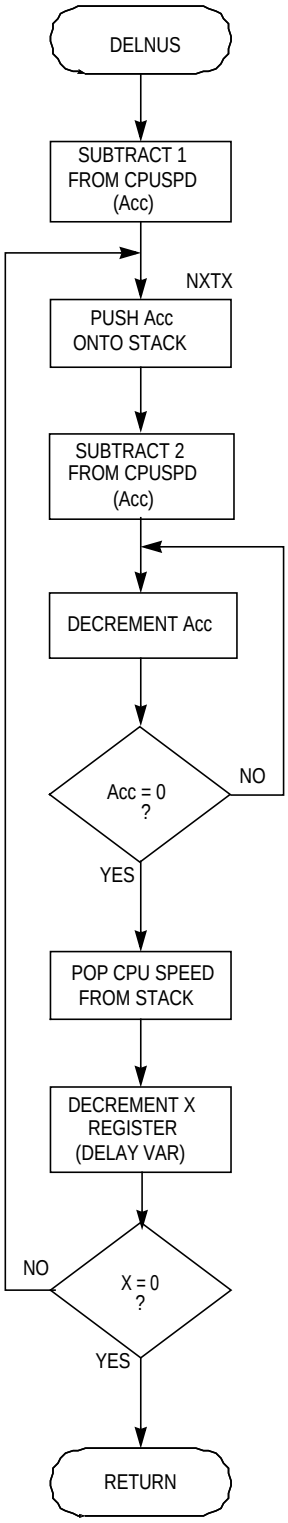


Figure 3. PRGRNGE



PURPOSE: DELAY FOR N*12 US FOR f_{OP} >= 1 MHZ;
D = (DELAY TIME[US]/12) IN X, C = (f_{OP}[MHZ]*4)
IN Acc CYCLES = 5+(DELAY/12)* 3($4f_{OP}$-3)+9 =
5+DELAY*f_{OP}

X CONTAINS THE TIME/12
OF DELAY (IN MICROSECONDS.);
Acc CONTAINS CPUSPD (CPU SPEED X 4);
CPU SPEED MUST BE >= 1 MHZ

Figure 4. DELNUS

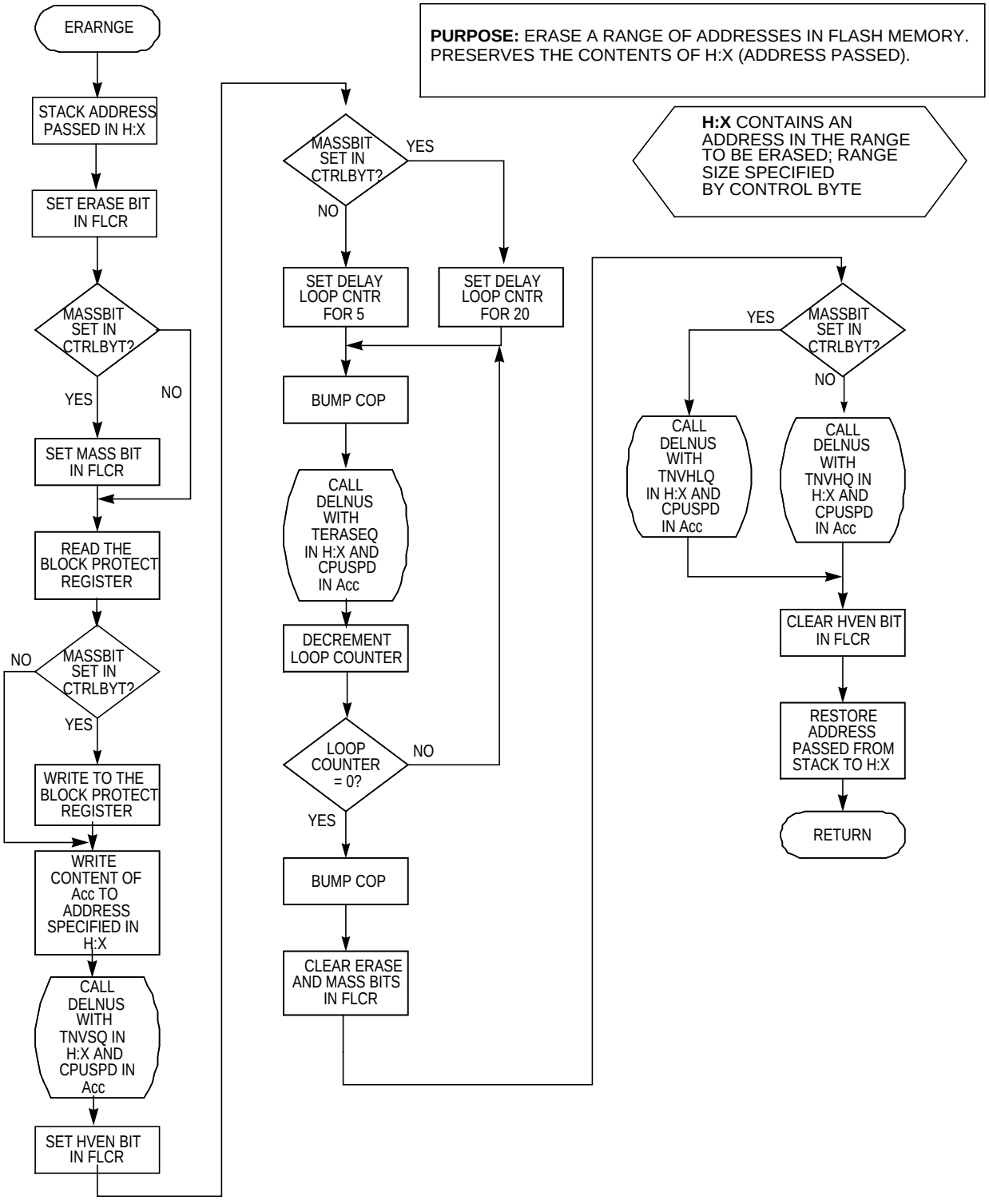


Figure 5. ERARNGE

ROM Routines Source Code

```
*******************************************************************************
* FILE NAME: MAINPR.ASM
* PURPOSE: To provide FLASH erase, program and verify routines
*          to reside in ROM.
* TARGET DEVICE: MC68HC908GR8, MC68HC908KX8, MC68HC908JL3/JK3 and the MC68HC908JB8
*
* MEMORY USAGE - RAM: 4-36 BYTES, DEPENDING ON DATA PASSED
*                ROM: 364 BYTES
*
* ASSEMBLER: MCUEZ
* VERSION: 1.0.5
*
* PROGRAM DESCRIPTION:
* This program contains a structure of routines to facilitate FLASH programming.
* These routines, which are individually callable, are intended to reside in ROM
* for the use of a user program, a test/burn-in program, or for development/programming
* tools. This set of routines is included, along with definition files, by the project
* file 9GR8ALLROM.ASM.
*
* AUTHOR: Grant Whitacre
* LOCATION: Austin - Oak Hill, Texas
*
* UPDATE HISTORY:
* REV    AUTHOR          DATE            DESCRIPTION OF CHANGE
* ===   ============    ========        ====================
* 0.0   G. WHITACRE     10/05/98        Initial release
* 0.1   G. WHITACRE     02/17/99        MODIFIED FOR THE SST FLASH
* 0.2   G. WHITACRE     08/23/99        MODIFIED GETBYTE FOR 9600
*                                       BAUD @ 2.4576 MHZ
*
* GENERAL CODING NOTES:
* Bit names are labeled with <port name><bit number> and are used in the commands that
* operate on individual bits, such as BSET and BCLR. A bit name followed by a dot
* indicates a label that will be used to form a bit mask.
*******************************************************************************
*******************************************************************************
* INCLUDED FILES
*******************************************************************************
*       INCLUDE "E:\MMDS\GR8\SSTROM\H908GR8.FRK"
*******************************************************************************
* EQUATES
*******************************************************************************
* PROGRAMMING TIMES IN µs
* FOLLOWING DEFINED IN .FRK FILE
*TPROG         EQU     40                  ;FLASH Byte Program Time
*TERASE        EQU     1000                ;FLASH Page Erase Time
*TMERASE       EQU     4000                ;FLASH Mass Erase Time
```

```
*TNVS          EQU      10                  ;FLASH PGM/ERASE to HVEN Setup Time
*TPGS          EQU      5                   ;FLASH Program Hold Time
*TNVH          EQU      5                   ;FLASH High-Voltage Hold Time
*TNVHL         EQU      100                 ;FLASH High-Voltage Hold Time (Mass Erase)
*TRCV          EQU      1                   ;FLASH Return to Read Time

* TIMES REPRESENT VALUES THAT ARE PASSED TO THE DELAY ROUTINE, WHICH
* DELAYS FOR X 12 µs FOR VALUES PASSED. FOR TERASE AND TMERASE, THE
* ROUTINE IS CALLED 5 AND 20 (12 µs*17*20=4080 µs) TIMES,
* RESPECTIVELY, WITH A BUMP OF THE COP BEFORE EACH CALL
ECALLS         EQU      5
MECALLS        EQU      20
TPROGQ         EQU      3                   ;FLASH Program Time
TERASEQ        EQU      17                  ;FLASH Block Erase Time
TMERASEQ       EQU      17                  ;FLASH Mass Erase Time
TNVSQ          EQU      1                   ;FLASH PGM/ERASE to HVEN Setup Time
TPGSQ          EQU      1                   ;FLASH Program Hold Time
TNVHQ          EQU      1                   ;FLASH High-Voltage Hold Time
TNVHLQ         EQU      8                   ;FLASH High-Voltage Hold Time (Mass Erase)
TRCVQ          EQU      1                   ;FLASH Return to Read Time

*****************************************************************************
* ROUTINES
*****************************************************************************
*****************************************************************************
* NAME: GETBYTE
* PURPOSE: Get a byte of data on PTA0
* Entry Conditions: Port A0 configured as an input.
* Exit Conditions:  Acc=byte received.
*                   If break received or result bad then send break and
*               jump back to start.
*                Port A0 configured as an input.
* SUBROUTINES CALLED: GET_BIT
* VARIABLES READ:
* VARIABLES MODIFIED:
* STACK USED: 4
* SIZE: 20 BYTES
* DESCRIPTION: EXECUTED OUT OF ROM
* Attempts to receive a byte from the external controller via PortA0.
* Once called, program will remain in GETBYTE until a byte is received
* Signal to start receiving a byte is a valid (low) start bit.
* NOTE: Cycle path for each bit reception must be kept the same to maintain
* a steady baud rate.
*BIT TIMING=9+(17+10*23)=256 CYCLES@2.4576 MHZ=104 µs=9600 BAUD
*****************************************************************************
GETBYTE:
       BRSET0 ,PTA,GETBYTE                 ;Waiting for start edge.
       JSR    GET_BIT                      ;try to receive a full start bit.
       BCS    GETBYTE                      ;Success?
       LDA    #$80                         ;initialize receiver.
GBIT:                                      ;got start bit, now get byte.
```

```
       JSR    GET_BIT                      ;5
       RORA                                ;1 bit into Acc
       BCC    GBIT                         ;3 get next bit
*                                          ;baud calculation
STOPBIT:
       JSR    GET_BIT                      ;look for stop bit
       RTS
*******************************************************************************
*******************************************************************************
* NAME: RDVRRNG
* PURPOSE: Read and/or Verify a range of FLASH memory
* ENTRY CONDITIONS: H:X contains the first address of the range;
*                   LADDR contain the last address to be read;
*                   Acc contains a Boolean to see if read data
*                   goes to PTA0 (0=PTA0, else Data Array)
*                   DATA contains the data to compare the read data against
* EXIT CONDITIONS: C bit is set if good compare; Acc contains checksum;
*                 DATA contains read FLASH data
* SUBROUTINES CALLED:
* VARIABLES READ: LADDR, DATA ARRAY
* VARIABLES MODIFIED: DATA ARRAY
* STACK USED: 6
* SIZE: 63 BYTES
* DESCRIPTION: EXECUTED OUT OF ROM; ALTHOUGH THIS ROUTINE SERVICES THE COP,
* THERE COULD STILL BE A COP TIME OUT UNDER CERTAIN CONDITIONS. THESE CONDITIONS
* ARE: 1) IN USER MODE, 2) COP ENABLED, 3) USING THE SHORT COP TIMEOUT, 4) NOT USING
* THE PLL SUCH THAT fOP = CGMXCLK/4
*******************************************************************************
RDVRRNG:
       PSHA                              ;(A)SAVE DESTINATION FLAG ON STACK AS 4,SP
       CLRA                              ;LOCAL VARIABLE FOR CHECKSUM STARTS AT 00
       PSHA                              ;(B)SAVE ON STACK AS 3,SP
                                         ;LOCAL VARIABL FOR INDEX INTO DATA STARTS AT 00
       PSHA                              ;(C)SAVE ON STACK AS 2,SP
       COMA                              ;LOCAL VARIABLE FOR VERIFY STATUS (FF = GOOD)
       PSHA                              ;(D)SAVE ON STACK AS 1,SP
RDVRRNG010:
       STA    $FFFF                      ;BUMP THE COP
       LDA    ,X                         ;LOAD CONTENT OF FLASH ADDRESS INTO ACC.
       TST    4,SP                       ;CHECK DESTINATION FLAG
       BEQ    RDVRRNG020                 ;SKIP COMPARE IF DESTINATION IS PTA0
       PSHX                              ;(E)STORE FADDR FOR LATER
       PSHH                              ;(F)
       LDX    4,SP                       ;GET INDEX INTO DATA FROM STACK
       CLRH
       CMP    DATA,X                     ;COMPARE ADDR NOW IN X SO COMPARE CONTENT
       BEQ    RDVRRNG015                 ;IF EQUAL THEN KEEP GOING...
       STA    DATA,X                     ;WRITE FLASH DATA THAT IS DIFFERENT TO RAM
       LDX    #$7E                       ;FAILED VERIFICATION SO CLEAR VERIFY STATUS
       STX    3,SP                       ;MUST KEEP DATA IN ACC FOR CHECKSUM BELOW
RDVRRNG015:
```

```
        PULH                            ;(F')GET FADDR BACK
        PULX                            ;(E')
        BRA     RDVRRNG030
RDVRRNG020:                             ;NOT COMPARING, JUST DUMPING
        JSR     PUT_BYTE                ;WRITE DATA TO PORT A0...
                                        ;PUT_BYTE SAVES A, X, AND H
RDVRRNG030:
        ADD     3,SP                    ;ADD VALUE OF CURRENT BYTE TO CHECKSUM
        STA     3,SP                    ;MAINTAIN AS RUNNING SUM
        INC     2,SP                    ;INCREMENT INDEX INTO DATA
        CPHX    LADDR                   ;COMPARE SOURCE ADDR TO THE LAST ADDRESS
        BHS     NOMO                    ;IF NOT YET DONE, LOOP FOR ANOTHER
        AIX     #1                      ;INCREMENT SOURCE ADDRESS
        BRA     RDVRRNG010
NOMO    PULA                            ;(D')GET PASS/FAIL INFO INTO
        TAP                             ; CARRY BIT
        PULA                            ;(C')TRASH INDEX INTO DATA
        PULA                            ;(B')RETURN CHECKSUM IN ACC.
        AIS     #1                      ;(A')TRASH DESTINATION FLAG
        RTS
*******************************************************************************
*******************************************************************************
* NAME: PRGRNGE
* PURPOSE: Programs a range of addresses in FLASH memory
* ENTRY CONDITIONS: H:X contains THE FIRST address in the range;
*           CTRLBYT contains the Control Byte that specifies
*           the programming mode; LADDR contains the last address
*           to be read; DATA contains the data to be programmed
* EXIT CONDITIONS: Next address in H:X
* SUBROUTINES CALLED: DELNUS
* VARIABLES READ: CONTROL BYTE, CPUSPD, LADDR, DATA ARRAY
* VARIABLES MODIFIED:
* SIZE: 170 BYTES
* STACK SIZE (INCLUDING CALL): 7 BYTES
* DESCRIPTION: EXECUTED OUT OF ROM
* Allows passing of a range of addresses to PRGRNGE, which does not have
* to be on row boundaries, either beginning or end. I.e., passing $F001 to
* $F008 is valid. This is to prevent trying to program a non-FLASH address.
*******************************************************************************
PRGRNGE:
        SEI                             ;MASK INTERRUPTS SO THAT DELAYS ARE NOT
                                        ; AFFECTED
        CLRA                            ;STORES INDEX INTO DATA ARRAY
        PSHA                            ;(A) INDEX INTO DATA IS ON STACK
        PSHX                            ;(B)SAVE FADDR SO THAT IT IS NOT DESTROYED
        PSHH                            ;(C)
        TXA                             ;GET (FADDR MODULUS ROWSIZE)
        LDX     #ROWSIZ
        CLRH                            ;HIGH BYTE CAN BE IGNORED BECAUSE ROWSIZE
                                        ; IS ALWAYS A POWER OF TWO AND 256 OR LESS.
                                        ; IT MUST BE IGNORED SO THAT RESULT OF DIVIDE
```

```
                                        ; WILL FIT IN ONE BYTE.
        DIV                             ;DIVIDE LEAVES REMAINDER (MODULUS) IN H
        PSHH                            ;(D)PUSH REMAINDER IN H ONTO STACK
        TXA                             ;MOVE ROWSIZE TO ACC
        SUB     1,SP                    ;SUBTRACT REMAINDER TO GET #BYTES TO PROGRAM
        PULH                            ;(D')PULL REMAINDER FROM STACK AND THROW AWAY
        PULH                            ;(C')GET FADDR BACK FROM STACK
        PULX                            ;(B')
        PSHA                            ;(B)STORE #BYTES TO END OF ROW ON STACK
        PSHA                            ;(C) RESERVE A STACK LOC. FOR COP LOOPING VAR.
                                        ;3,SP = COP LOOPING VARIABLE
                                        ;4,SP = #BYTES TO END OF ROW
                                        ;5,SP = INDEX INTO DATA ARRAY
PRGSTP1:
        STA     $FFFF                   ;BUMP COP
        LDA     #$06                    ;SET LOOPING VARIABLE TO ALLOW FOR COP BUMP;
        STA     1,SP                    ;NEED TO TURN OFF PGM AND HVEN OCCASIONALLY TO
                                        ; BUMP COP
         LDA    #PGM.                   ;SET PGM BIT
         ORA    FLCR
         AND    #$F9                    ;($FF-MERASE.-ERASE.)
                                        ;MAKE SURE ERASE BITS ARE OFF
        STA     FLCR                    ;WRITE THIS TO THE FLASH CONTROL REG.

PRGSTP2 LDA     FLBPR                   ;READ FROM BLOCK PROT. REG.

PRGSTP3:
        IFEQ    TESTMOD
        LDA     ,X
        ENDIF
        IFNE    TESTMOD
        STA     ,X                      ;WRITE TO ANY FLASH ADDRESS WITHIN THE ROW
        ENDIF
                                        ;TO BE PROGRAMMED WITH ANY DATA
        PSHH                            ;(D)
        PSHX                            ;(E)

PRGSTP4 LDX     #TNVSQ                  ;DELAY FOR TNVS
        LDA     CPUSPD
        BSR     DELNUS

PRGSTP5 LDHX   #FLCR                    ;SET THE HVEN BIT IN FLCR
        LDA     ,X
        ORA     #HVEN.
        STA     ,X

PRGSTP6 LDX     #TPGSQ                  ;DELAY FOR TIME TPGS
        LDA     CPUSPD
        BSR     DELNUS

        PULX                            ;(E')
        PULH                            ;(D'
```

```
*******************************************************************************
* NEED TO PROGRAM 6 BYTES, TURN OFF PGM AND/OR HVEN, BUMP COP, PROGRAM ANOTHER
* 6 BYTES, THEN REPEAT PROCESS UNTIL FINISHED WITH RANGE
*******************************************************************************
PRGSTP7 PSHH                                ;(D)
        PSHX                                ;(E)
                                            ;1,SP = ADDR(LSB)
                                            ;2,SP = ADDR(MSB)
                                            ;3,SP = COP LOOPING VARIABLE
                                            ;4,SP = #BYTES TO END OF ROW
                                            ;5,SP = INDEX INTO DATA ARRAY
        CLRH                                ;GET 0:BUFFPTR INTO H:X
        LDX     5,SP                        ;GET THE INDEX INTO DATA ARRAY
        LDA     DATA,X                      ;LOAD BYTE TO PROG FROM DATA+BUFFPTR
        PULX                                ;(E') POP LO BYTE OF ADDR BACK INTO X
        PULH                                ;(D')
    IFEQ TESTMOD
        LDA     ,X
    ENDIF
    IFNE TESTMOD
        STA     ,X                          ;STORE DATA TO ADDR SPEC.BY H-X
    ENDIF
        PSHH                                ;(D)
        PSHX                                ;(E)
PRGSTP8 LDX     #TPROGQ                     ;DELAY FOR TPROG
        LDA     CPUSPD
        BSR     DELNUS
        PULX                                ;(E')
        PULH                                ;(D')

PRGSTP9:
        AIX     #$01                        ;INCREMENT THE DESTINATION ADDRESS
        INC     3,SP                        ;INCREMENT THE POINTER INTO DATA
        DEC     2,SP                        ;DECREMENT THE BYTE COUNTER
        DEC     1,SP                        ;DECREMENT COP LOOPING VARIABLE
        CPHX    LADDR                       ;CHECK FOR END OF RANGE
        BHI     PRGSTP10                    ;EXIT LOOP IF PAST END OF RANGE
        TST     2,SP                        ;CHECK FOR END OF ROW
        BEQ     PRGSTP10                    ;EXIT LOOP IF DONE WITH ROW
        TST     1,SP
        BNE     PRGSTP7                     ;COP VAR = 0?
        BSR     CLR_P_H                     ;
        TAX
        BRA     PRGSTP1                     ;

PRGSTP10:
        BSR     CLR_P_H                     ;CALL RTN TO CLEAR PGM AND HVEN
NEXTROW:                                    ;DONE WITH ROW, GET READY TO EXIT
                                            ;1,SP = COP LOOPING VARIABLE
                                            ;2,SP = #BYTES TO END OF ROW
                                            ;3,SP = INDEX INTO DATA ARRAY
```

```
        ADD     2,SP                            ;ADD BYTES PROGRAMMED TO LOW BYTE
        TAX
        PSHH                                    ;(D) CORRECT HIGH BYTE FOR CARRY, IF ANY
        PULA                                    ;(D')
        ADC     #0
        PSHA                                    ;(D)
        PULH                                    ;(D')

        LDA     #ROWSIZ                         ;
        STA     2,SP                            ;#BYTES TO END OF ROW IS ROWSIZE
        AIX     #-1                             ;DECREMENT CURRENT ADDRESS BY 1 TO COMP.
                                                ; TO LAST ADDR
        CPHX    LADDR                           ;COMPARE FADDR TO LADDR
        AIX     #1
        BLO     PRGSTP1                         ;PROGRAM ANOTHER ROW IF LESS OR EQUAL

PRGSTP13:                                       ;NEXT 3 INST. TAKE > 1 µs.
        PULA                                    ;(C')REMOVE COP LOOP VARIABLE
        PULA                                    ;(B')REMOVE #BYTES TO END OF ROW
        PULA                                    ;(A')REMOVE INDEX INTO DATA ADDRESS

DONEPRG RTS


* FOLLOWING LOCAL SUB-ROUTINE CLEARS PGM, DELAYS, THEN CLEARS HVEN.
CLR_P_H PSHH                                    ;(D)
        PSHX                                    ;(E)
        LDHX    #FLCR                           ;CLEAR PGM BIT
        LDA     ,X
        EOR     #PGM.
        STA     ,X
PRGSTP11:
        LDX     #TNVHQ                          ;DELAY FOR TNVH
        LDA     CPUSPD
        BSR     DELNUS
PRGSTP12:
        LDHX    #FLCR                           ;CLEAR THE HVEN BIT
        LDA     ,X
        EOR     #HVEN.
        STA     ,X
        PULA                                    ;(E')
        PULH                                    ;(D')
        RTS
*******************************************************************************
```

```
*******************************************************************************
* NAME: DELNUS
* PURPOSE: Delay N ms
* ENTRY CONDITIONS: X CONTAINS THE TIME/12 OF DELAY (IN ms).
*           A CONTAINS THE CPU SPEED X 4 (2 BITS OF PRECISION)
* EXIT CONDITIONS:
* SUBROUTINES CALLED:
* VARIABLES READ:
* VARIABLES MODIFIED:
* SIZE: 10 BYTES
* STACK USED (INCLUDING CALL): 3 BYTES
* DESCRIPTION: EXECUTED OUT OF ROM
* Delay Routine for fOP >= 1 MHz, Delay >= 12 ms
* (delay time[µs]/12) in H:X, (fOP[MHz]*4) in Acc
* If fOP > 1 then
* CYCLES = 5+Delay/12[3(4fOP-3)+9] = 5+DELAY*fOP
* If fOP = 1 then CYCLES = 5+12(DELAY/12) = 5+DELAY
* where delay in µs and fOP in MHz
*******************************************************************************
DELNUS: DECA                                ;1 CYCLE
NXTX    PSHA                                ;2
        DECA                                ;1
        DECA                                ;1
        DBNZA   *                           ;3
        PULA                                ;2
        DBNZX   NXTX                        ;3
        RTS                                 ;4
*******************************************************************************
*******************************************************************************
* NAME: ERARNGE
* PURPOSE: Erase a range of addresses in FLASH memory
* ENTRY CONDITIONS: H-X contains an address in the range to be erased; range size
*             specified by Control Byte
*             If b6 = 1 then mass erase, otherwise erase
*             1 page (64 bytes for the GR8).
* EXIT CONDITIONS: Preserves the contents of H:X (address passed)
* SUBROUTINES CALLED: DELNUS
* VARIABLES READ: CTRLBYT, CPUSPD
* VARIABLES MODIFIED:
* STACK USED: 5
* SIZE: 99 BYTES
* DESCRIPTION: Does not check for a blank range before (to see if erase
*              is necessary) or after (to see if successful erase)
*******************************************************************************
ERARNGE:
        SEI
        PSHH                                ;KEEP ADDRESS PASSED
        PSHX

        CLRA                                ;SET ERASE BIT, AND
        ORA     #ERASE.
```

```
        BRCLR  MASSBIT,CTRLBYT,AMBS
        ORA    #MASS.                       ;MASS BIT IF NECESSARY
AMBS:   STA    FLCR
ERABLK  LDA    FLBPR                        ;READ THE BLOCK PROTECT REGISTER
    IFEQ TESTMOD                            ;WRITE TO ANY ADDRESS IN ERASE RANGE
        LDA    FLBPR
        LDA    ,X
    ENDIF
    IFNE TESTMOD
        BRCLR  MASSBIT,CTRLBYT,NOBLWR
        STA    FLBPR
NOBLWR  STA    ,X
    ENDIF

        LDX    #TNVSQ                       ;DELAY FOR TNVS
        LDA    CPUSPD
        BSR    DELNUS

        LDHX   #FLCR                        ;SET THE HVEN BIT IN FLCR
        LDA    ,X
        ORA    #HVEN.
        STA    ,X

        BRCLR  MASSBIT,CTRLBYT,RWERASE
        LDA    #MECALLS                     ;DELAY LOOPS FOR TMERASE
        BRA    ERADEL                       ;       OR
RWERASE LDA    #ECALLS                      ;DELAY LOOPS FOR TERASE

ERADEL  PSHA                                ;STACK INCREMENT COUNTER
BUMPCOP STA    $FFFF                        ;BUMP COP
        LDX    #TERASEQ                     ;SAME FOR TERASEQ AND TMERASEQ
        LDA    CPUSPD
        BSR    DELNUS
        DEC    1,SP
        BNE    BUMPCOP
        PULA                                ;PULL INCREMENT CNTR OFF STACK
        STA    $FFFF                        ;BUMP COP WHEN DONE DELAYING
        LDHX   #FLCR                        ;CLEAR THE ERASE BIT

        LDA    ,X
        EOR    #ERASE.
        AND    #($FF-MASS.)                 ;CLEAR MASS BIT
        STA    ,X

        BRCLR  MASSBIT,CTRLBYT,PGSTUP
        LDHX   #TNVHLQ                      ;DELAY FOR TNVHL
        BRA    STUPDEL                      ;       OR
PGSTUP  LDHX   #TNVHQ                       ;DELAY FOR TNVH
STUPDEL LDA    CPUSPD
         BSR   DELNUS
```

```
        LDHX    #FLCR                           ;CLEAR THE HVEN BIT
        LDA     ,X
        EOR     #HVEN.
        STA     ,X

XERARNG PULX                                    ;RESTORE ADDRESS PASSED
        PULH                                    ;THESE 3 INST. DELAY FOR
        RTS                                     ;AT LEAST 1 µs (TRCV)
*******************************************************************************
```

NOTE: *The following routines are resident in the MC68HC908KX8 only.*

```
*******************************************************************************
* ROUTINE NAME: ICGTRIM
* PURPOSE: AN ICG TRIM ROUTINE BASED ON THE MEASUREMENT OF THE
* LENGTH OF A BREAK SIGNAL SENSED ON PTA0 OR PTB4/RXD.
* ENTRY CONDITIONS:    ICG IS ENABLED (ICGON IS SET); INTERNAL CLOCK IS SELECTED
*                      (CS IS CLEARED); ACC IS CLEARED TO SELECT PTB4/RxD TO MONITOR
*                      BREAK SIGNAL, ACC IS NON-ZERO TO SELECT PTA0; PORT USED HAS
*                      BEEN CONFIGURED IN SW AS AN INPUT AND IN HW FOR NRZ
*                      COMMUNICATION.
* EXIT CONDITIONS:     CARRY BIT IS SET IF ICG WAS TRIMMED SUCCESSFULLY;
*                      MONITOR PORT CONFIGURED AS AN INPUT
* SUBROUTINES CALLED: NONE
* VARIABLES READ: PTA OR PTB
* VARIABLES MODIFIED: ICGTR, ICGCR, ICGMR
* STACK USED: 1 BYTE
* SIZE: 67 BYTES
* DESCRIPTION: EXECUTED OUT OF ROM, THIS ROUTINE CHECKS TO SEE HOW
* MANY CYCLES ARE MEASURED DURING A BREAK SIGNAL (10 LOW BITS)
* SENT AT 9600 BAUD BY A HOST AND ADJUSTS ITS TRIM AND MULTIPLIER
* REGISTERS. IF THE BREAK SIGNAL IS MORE THAN 25% VARIATION FROM
* WHAT IS EXPECTED (.78-1.30 µs @ 9600), THEN ICG TRIMMING WILL
* NOT BE PERFORMED. THIS ICG ACCURACY LIMIT IS CONSISTENT WITH
* THE EXTENT OF THE ICG'S ABILITY TO FINE-TUNE THE TRIM REGISTER.
*******************************************************************************
ICGTRIM:
 MOV    #$20,ICGMR                              ;SET ICG TO 307.2 KHZ * 32 = 9.8304 MHZ
        BRCLR   ICGS,ICGCR,*                    ;WAIT FOR CLOCK TO STABILIZE
        CLRX
        CLRH
        TSTA                                    ;SEE IF PTA0 OR PTB4 IS USED
        BEQ     MONPTB4                         ;BRANCH IF BLANK TO MONITOR PTB4
        BRSET   0,PTA,*                         ;WAIT FOR BREAK SIGNAL TO START

* FOLLOWING LOOP IS EXECUTED UNTIL THE END OF THE BREAK SIGNAL. THE BREAK
* SIGNAL LASTS 10 BIT TIMES. IF COMMUNICATING AT fOP/256 BPS, THEN 10 BIT
* TIMES IS 2560 CYCLES. EACH TIME THROUGH THE LOOP IS 10 CYCLES, SO WE
* EXPECT TO EXECUTE THE LOOP 256 TIMES IF THE KX8 IS IN SYNC SERIALLY WITH
* THE HOST. IF WE STAY IN THE LOOP FOR > 256 LOOP CYCLES, THEN THE KX8
* MUST BE RUNNING FASTER THAN EXPECTED, AND NEEDS TO BE SLOWED DOWN. IF WE
```

```
* STAY IN THE LOOP FOR < 256 LOOP CYCLES THEN THE KX8 MUST BE RUNNING SLOWER
* THAN EXPECTED AND NEEDS TO BE SPEEDED UP. THE AMOUNT THAT WE CHANGE THE
* CPU SPEED IS EQUAL TO THE NUMBER OF LOOP CYCLES OVER OR UNDER 256. SO IF
* WE GO THROUGH THE LOOP 240 TIMES, THEN WE ARE RUNNING
* (256-240)/256 = 6.25% FAST. EACH INCREMENTAL CHANGE WE MAKE TO THE TRIM REGISTER
* (ICGTR) WILL MAKE A 0.195% CHANGE TO THE INTERNAL CLOCK. THAT IS, INCREMENTING
* THE REGISTER BY ONE OVER THE DEFAULT VALUE OF $80 STORED THERE WILL
* DECREASE THE INTERNAL CLOCK BY 0.195%, AND VICE VERSA.
* NOW EACH EXECUTION OF THE LOOP OVER OR UNDER WHAT IS EXPECTED (256 TIMES)
* REPRESENTS AN ERROR OF 1/256 = .391% ERROR. SO WE'LL NEED TO DOUBLE THE
* NUMBER OF LOOP CYCLES AND USE THIS NUMBER TO CORRECT THE TRIM REGISTER.
* OUR PRECISION FOR TRIMMING IS THEREFORE 0.391%.
*
* COUNTS RECEIVED AT DEVICE BAUD RATE OF 9600 (fOP = 2.4576 MHZ):
* BAUD RATE  EXPECTED COUNT     MIN COUNTS      MAX COUNTS     ICGMR VAL
* =========  ==============     ==========      ==========     =========
*       9600   256 (0100H)         192 (00C0H)     320 (0140H)      $20

CHKPTA0 BRSET  0,PTA,BRKDONE       ;(5) GET OUT OF LOOP IF BREAK IS OVER
        AIX    #1                  ;(2) INCREMENT THE COUNTER
        BRA    CHKPTA0             ;(3) GO BACK AND CHECK SIGNAL AGAIN

MONPTB4 BRSET  4,PTB,*             ;WAIT FOR BREAK SIGNAL TO START
CHKPTB4 BRSET  4,PTB,BRKDONE       ;(5) GET OUT OF LOOP IF BREAK IS OVER
         AIX   #1                  ;(2) INCREMENT THE COUNTER
         BRA   CHKPTB4             ;(3) GO BACK AND CHECK SIGNAL AGAIN

BRKDONE PSHH
        PULA                       ;PUT HIGH BYTE IN ACC AND WORK WITH A:X
        TSTA                       ;IF MSB OF LOOP CYCLES = 0, THEN BREAK TAKES TOO
        TXA                        ;FEW CYCLES THAN EXPECTED, SO TRIM BY SPEEDING
        BEQ    SLOW                ;UP fOP.
FAST    CMP    #$40                ;SEE IF BREAK IS WITHIN TOLERANCE
        BGE    OOR                 ;DON'T TRIM IF OUT OF RANGE
        ADD    #$80                ;BREAK LONGER THAN EXPECTED, SO SLOW DOWN fOP
        BRA    ICGDONE
SLOW    CMP    #$C0                ;SEE IF BREAK IS WITHIN TOLERANCE
        BLT    OOR                 ;DON'T TRIM IF OUT OF RANGE
        SUB    #$80
ICGDONE STA    ICGTR
    IFEQ TESTMOD
        BSR    ICGTEST
    ENDIF
EXITTRM SEC                        ;SET CARRY SIGNIFYING TRIM OCCURRED
        RTS
OOR     CLC                        ;CLEAR CARRY SIGNIFYING NOT TRIMMED
        RTS
```

```
*******************************************************************************
* NAME: ICGTEST
* PURPOSE: Following tests the above ICG settings to see if the internal clock is set
* at the desired rate. Internal clock rate is 16x frequency sensed at bit 4 of port A.
* ENTRY CONDITIONS: NONE
* EXIT CONDITIONS: IRQ PULLED LOW TO EXIT, PTA4 SET AS OUTPUT
* SUBROUTINES CALLED: NONE
* VARIABLES READ:
* VARIABLES MODIFIED: PTA, DDRA
* STACK USED: 0
* SIZE: 13 BYTES
* DESCRIPTION: EXECUTED OUT OF ROM
*******************************************************************************
ICGTEST BSET   4,DDRA                       ;bit 1 set as output
BITOFF  BCLR   4,PTA                        ;4 cycles
        BIL    EXITLP                       ;3 cycles
        NOP                                 ;1 cycle
BITON   BSET   4,PTA                        ;4 cycles
        NOP                                 ;1 cycle
        BRA    BITOFF                       ;3 cycles
EXITLP  RTS                                 ;16 cycles
*******************************************************************************
```

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. Customer Focus Center, 1-800-521-6274

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

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

Mfax™, Motorola Fax Back System: RMFAX0@email.sps.mot.com; http://sps.motorola.com/mfax/; TOUCHTONE, 1-602-244-6609; US and Canada ONLY, 1-800-774-1848

HOME PAGE: http://motorola.com/sps/