

Motorola Semiconductor Application Note

AN1770

In-Circuit Programming of FLASH Memory in the MC68HC908GP20

By Grant Whitacre
Microcontroller Division
Austin, Texas

Introduction

This application note describes two methods of programming FLASH memory in the Motorola MC68HC908GP20 (GP20) microcontroller, a general-purpose device based on the 68HC08 architecture that has 20 Kbytes of on-board FLASH.

Programming the FLASH array can be done either in user mode or monitor mode. The information given here details:

- How the FLASH is programmed and erased in-circuit in each of these two modes
- How the control and protection registers are programmed
- Additional considerations when dealing with this type of memory

To illustrate the GP20's in-circuit programming capabilities, a sample program is included which executes programming routines from RAM.



These RAM routines are loaded in one of two ways:

- When programming in user mode, the routines would be part of and transferred into RAM by a bigger program residing in FLASH. This program could be initiated by the user's main program, perhaps through the monitoring of an input port, or it could be loaded into FLASH as a stand-alone seed program which would later enable re-programming of the entire FLASH array with the actual user program.
- Alternatively, the RAM routines could be loaded directly into RAM by an external host with the device in monitor mode.

The main routine of this dual mode program monitors the SCI (serial communications interface) port when user mode programming, or port A bit 0 when monitor mode programming, for the input of data and the address range in FLASH where the data is to be programmed. It then makes necessary calls to other RAM-loaded routines to do necessary erasing, programming, and verifying.

A host program's necessary functionality is also described in this application note. The one specifically used in the generation of this document is a Windows® application and accommodates both modes of programming. It is available as a free download from the Motorola web site, <http://mot-sps.com/csic/techhelp/appsw/appsw.htm>.

The user will be able to in-circuit program the GP20 in either monitor mode or user mode with:

- This host program
- Two MC68HC908GP20 RAM programs in S19 format
- A target system that is configured to communicate with a PC host
- A basic understanding of the device and its FLASH memory

Windows is a registered trademark of Microsoft in the U.S. and other countries.

Description of FLASH Memory in the MC68HC908GP20

Memory Map FLASH Location

The memory map for the MC68HC908GP20 is shown in **Figure 1**. Note that the FLASH memory occupies addresses from \$B000 to \$FDFF, a single byte for the block protection register at \$FF80, and a block for the user vectors from \$FFDC to \$FFFF. The total addressable FLASH capacity is 20,005 bytes.

\$0000 ↓	I/O Registers 64 Bytes
\$003F ↓	RAM 512 Bytes
\$023F ↓	Unimplemented 44,480 Bytes
\$0240 ↓	
\$AFFF ↓	FLASH Memory 19,968 Bytes
\$B000 ↓	
\$FDFF	
\$FE00	SIM Break Status Register (SBSR)
\$FE01	SIM Reset Status Register (SRSR)
\$FE02	Reserved (SUBAR)
\$FE03	SIM Break Flag Control Register (SBFCR)
\$FE04	Interrupt Status Register 1 (INT1)
\$FE05	Interrupt Status Register 2 (INT2)
\$FE06	Interrupt Status Register 3 (INT3)
\$FE07	Reserved (FLTCR)
\$FE08	FLASH Control Register (FLCR)

Figure 1. Memory Map (Sheet 1 of 2)

\$FE09	Break Address Register High (BRKH)
\$FE0A	Break Address Register Low (BRKL)
\$FE0B	Break Status and Control Register (BRKSCR)
\$FE0C	LVI Status Register (LVISR)
\$FE0D	Unimplemented 3 Bytes
↓	
\$FE0F	
\$FE10	Unimplemented 16 Bytes Reserved for Compatibility with Monitor Code for A-Family Parts
↓	
\$FE1F	
\$FE20	Monitor ROM 307 Bytes
↓	
\$FF52	
\$FF53	Unimplemented 45 Bytes
↓	
\$FF7F	
\$FF80	FLASH Block Protect Register (FLBPR)
\$FF81	Unimplemented 91 Bytes
↓	
\$FFDB	
\$FFDC	FLASH Vectors 36 Bytes
↓	
\$FFFF	

Note: \$FFF6-\$FFFD
Also Used for
8 Security Bytes

Figure 1. Memory Map (Sheet 2 of 2)

FLASH Control Register

The FLASH control register is located at \$FE08 in the user memory map. This register provides the means to erase, program, and verify the FLASH. All bits in this register can be read or written at any time. The register structure is shown in [Figure 2](#).

Address: \$FE08

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

Figure 2. FLASH Control Register (FLCR)

The FDIV bits are frequency divide control bits and are set to ensure proper operation for the charge pump. Its optimum frequency is about 2 MHz and is derived by dividing the internal bus frequency by a value stored in FDIV1 and FDIV0. Setting of these two bits should be made according to bus frequency as shown in [Table 1](#).

Table 1. Charge Pump Clock Frequency as a Function of Bus Frequency

FDIV1	FDIV0	Pump Clock Frequency	Use When Bus Frequency Is
0	0	Bus frequency ÷ 1	1.8–2.5 MHz
0	1	Bus frequency ÷ 2	3.6–5.0 MHz
1	0	Bus frequency ÷ 2	3.6–5.0 MHz
1	1	Bus frequency ÷ 4	7.2–10.0 MHz

NOTE: *There is no mechanism to step up the pump clock frequency from a bus frequency lower than 1.8 MHz to attain a charge pump frequency of about 2 MHz. Also, programming and erasing the FLASH cannot be done if the charge pump frequency is lower than 1.8 MHz.*

BLK1 and BLK0 are block erase control bits and are used to specify the size and location of a block of FLASH to be erased. Erasing can take place in chunks of a single row (64 bytes), eight rows (512 bytes), 4 Kbytes, 16 Kbytes, or the entire array. The procedure for erasing any

of these blocks of FLASH is discussed in [Procedure for Erasing the FLASH](#). The setting of the BLK bits corresponds to the erase block as shown in [Table 2](#).

Table 2. Size of Erase Blocks

BLK1	BLK0	Block Size
0	0	Full array: 20 Kbytes
0	1	Partial array*: 4 or 16 Kbytes
1	0	Eight rows: 512 bytes
1	1	Single row: 64 bytes

* \$B000–\$BFFF or \$C000–\$FFFF, depending on bit A14 of address written to

HVEN is the high-voltage enable bit and provides the control to apply the charge pump voltage to the cells to be erased or programmed.

The MARGIN bit is used during programming to verify that the page attempting to be programmed gets programmed adequately. It can only be set for verification after the HVEN bit is turned off. If it is high when HVEN is turned on, it will clear automatically. Setting the MARGIN bit puts the FLASH in a “hard read” state, which ensures that if the FLASH is verified in this state then it guarantees a good read during normal operation.

The ERASE and PGM (program) control bits are required to be set when erasing and programming, respectively. Both cannot be set at the same time.

Block Protection

The block protection register provides a way of preventing a block of FLASH from being erased or programmed. This register, itself a FLASH address, has four bits with each bit protecting a progressively larger block of FLASH starting at \$FFFF. [Figure 3](#) shows the register contents and the FLASH range that is protected by setting each bit.

Erasing this register clears all bits and removes protection from all blocks.

NOTE: *Setting more than one bit is redundant, and three of the bits provide protection of a FLASH range some of whose locations are not available in the GP20. Setting bits 0, 1, or 2 protects all of FLASH, and setting only bit 3 protects all but the lower 1000H bytes.*

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	Bit 7	Bit 6	Bit 5	Bit 4	BPR3	BPR2	BPR1	BPR0
Write:								
Reset:	U	U	U	U	U	U	U	U

U = Unaffected by reset. Initial value from factory is 0.

Figure 3. FLASH Block Protect Register (FLBPR)

Table 3. Block Protection Register and Size and Location of Block Protected

Bit Location	Bit Name	Address Block Protected
0	BPR0	\$B000–\$FFFF
1	BPR1	\$B000–\$FFFF
2	BPR2	\$B000–\$FFFF
3	BPR3	\$C000–\$FFFF

Protection of any block protects the block protection register because it is located at address \$FF80. The only way to circumvent block protection to modify FLASH is to enter monitor mode with V_{TST} applied to the IRQ line upon reset. (See the monitor mode entry in [Circuit Requirements](#).) When this occurs, all of FLASH, including the block protection register, can be erased or programmed regardless of block protection.

NOTE: *Since the block protection register is located at \$FF80, it is the first byte in a row whose range is \$FF80–\$FFBF. There are no other implemented FLASH locations within this row, so this register can be erased with no impact to other locations by specifying row erasing with the BLK bits in the FLASH control register (BLK1 = BLK0 = 1).*

To program FLASH in user mode, make sure that the FLASH to be modified is not block protected; otherwise, turn block protection off as in the example described here.

Procedure for Erasing the FLASH

The FLASH array is erased as a block, the size of which is determined by the BLK bits of the FLASH control register, as shown in [Table 2](#). Follow this step-by-step procedure to erase a block of FLASH reliably.

1. Make sure that the block to be erased is not protected by the settings in the block protection register. See [Block Protection](#) for a description of clearing protection for a block.
2. Write to the FLASH control register with a bit pattern that sets the appropriate FDIV bits and BLK bits. Set the ERASE bit at the same time.
3. Read the block protect register so that the FLASH control logic can latch its content.
4. Write any data to any FLASH address within the block to be erased.
5. Set the HVEN bit in the FLASH control register to apply the programming (charge pump) voltage.
6. Delay for a time, t_{Erase} (FLASH erase time), while the programming voltage is applied. Consult the memory characteristics information in the electrical specifications section of *MC68HC908GP20 Advance Information*, Motorola document order number MC68HC908GP20/D, for this value and other times referenced in this application note.
7. Clear the HVEN bit to turn off the programming voltage.
8. Delay for a time, t_{Kill} (high voltage kill time), to allow the high voltage of the charge pump to dissipate.
9. Clear the ERASE bit.
10. Delay for a time, t_{HVD} (FLASH return to read time), before trying to read from this block of FLASH.

Optionally, an erase verification may be performed after step 10. This verification would be done with a normal read of each location of FLASH, and a location would be verified as erased if a value of 0 is read.

NOTE: *It is recommended that a row be erased after eight programs of any page/pages of the row.*

Procedure for Programming the FLASH

The FLASH array is programmed a page at a time, where a page is defined as eight contiguous bytes whose first byte is on an 8-byte boundary. The normal programming sequence includes a verification step during which the FLASH is put in a “margin read” verification mode. In this mode, the control gates are held at a lower voltage than in a normal read. To allow sensing of the lower cell current, reads in this mode last eight machine cycles longer than a normal read.

Follow this procedure to reliably program and verify a page of FLASH.

1. Make sure that the page to be programmed is not within a protected block. See the block protection section for a description of clearing protection for a block.
2. Write to the FLASH control register with a bit pattern that sets the appropriate FDIV bits. Set the PGM bit at the same time.
3. Read the block protect register so that the FLASH control logic can latch its content.
4. Write individual data to each of the eight bytes of the page to be programmed.
5. Set the HVEN bit in the FLASH control register to apply the programming (charge pump) voltage.
6. Delay for a time, t_{PROG} (FLASH program time), while the programming voltage is applied.
7. Clear the HVEN bit to turn off the programming voltage.
8. Delay for a time, t_{HVTV} (FLASH HVEN low to MARGIN high time), to allow enough time between applying high voltage and doing a verification.
9. Set the MARGIN bit to initiate verification mode.
10. Delay for a time, t_{VTP} (FLASH MARGIN high to PGM low time).
11. Clear the PGM bit.
12. Delay for a time, t_{HVD} (FLASH return to read time), before trying to read from this block of FLASH.

13. Read the individual data in each of the eight byte locations to compare to what was intended to be programmed.
14. Clear the MARGIN bit.
15. Repeat steps 2 through 14 for each page until the data matches.

Practical Considerations for Programming, Verifying, and Erasing

Life Expectancy in Terms of Program/Erase Cycling

The FLASH in the MC68HC908GP20 has an endurance specification of 100 erase/program cycles. Therefore, it is not suited for short-term or temporary non-volatile parameter storage. This limitation could be alleviated somewhat by devising a scheme to move the parameter storage area when it approaches the 100-cycle limit.

For example, if a row (64 bytes) of parameter data is to be stored and updated on a frequent basis and the program is short enough to occupy less than two-thirds of the 20-Kbyte capacity, then a method of moving the location of the parameter block when the number of erase/program cycles approach 100 (cycle count could be stored in this block) could be implemented. In this example, this would effectively increase the number of parameter updates to 10,400.

Unused FLASH: $1/3 \times 19,968 \text{ bytes} = 6656 \text{ bytes} = 104 \text{ rows}$

$100 \text{ cycles/row} \times 104 \text{ rows in the unused FLASH space} = 10,400$
erase/program cycles

Since the smallest block erased is an entire row, whenever one byte is changed, the entire row would need to be erased and reprogrammed. Because of this limitation with FLASH, using it to store non-volatile temporary values is of limited usefulness.

Margin Programming

Margin, or bump, programming is a method of FLASH programming which is done in successive periods of relatively short duration, such as 1 millisecond. After each “bump,” the page that is attempted to be programmed is checked in margin mode, and if verification of any byte in the page fails, the entire page is re-programmed, without erasing, for another bump period. This process is repeated until all bytes in the page pass verification.

Bump programming minimizes the programming time, not only for the user’s benefit but also for the endurance of the FLASH. By minimizing the amount of time that a FLASH cell is exposed to the programming voltage, the life of the cell is maximized and the possibility of disturbing (inadvertently programming) neighboring cells by an excessively long programming time is minimized. The RAM program contained in this application note utilizes this method of programming, as does the information in [Procedure for Programming the FLASH](#).

Programming the MC68HC908GP20 in User Mode

Program Algorithm

Included in this application note is a program that demonstrates the capability to perform in-circuit programming. By setting assembler directives, this program can be loaded into FLASH and run in user mode, so no special mode entry hardware is required. After an initialization and loading of the RAM routine executed from FLASH, initiated either upon reset or subroutine call from the user’s existing main program, the program branches to a place in RAM where program execution resumes.

From RAM, the SCI serial communication port is monitored for the download of data to be programmed into FLASH and the starting address to place this data. After this data is received, the main RAM routine calls necessary subroutines, also located in RAM, to perform necessary FLASH erasing, programming, verification, and memory dumping. Up to a row (64 bytes) of data can be downloaded at a time, although there are a few stipulations when downloading data to be programmed.

They are:

- All data downloaded must be intended to reside within a single row boundary. This means that the start address must be divisible by 64 when an entire row is to be programmed.
- If data forming less than an entire row is downloaded, the previous rule still applies. All intended addresses must be within a single row boundary. Additionally, since programming takes place eight bytes at a time, a multiple of eight bytes should be downloaded and the starting address must be on a page (8-byte) boundary.
- The program checks to see if all intended addresses are erased before attempting to program. If any cell is not erased, then the entire row is erased. So, if programming less than an entire row, remember that the entire row will be erased if the block designated for reprogramming is not completely erased.
- At the completion of each block programming, the data is read from the current row and all 64 bytes of the row are sent out the SCI port, even if less than the entire row was reprogrammed.

This main RAM routine executes a continuous loop so that multiple data downloads can take place without the program ever leaving RAM. In fact, the entire FLASH array can be (re)programmed in this manner. At the completion of programming, the device will need to be reset to take it out of the SCI monitoring routine and to execute the new code in FLASH.

After receiving data to be programmed and loading RAM with necessary programming code, program execution jumps to RAM to do the actual FLASH programming.

The program consists of the functions listed here:

- Initialize all variables, ports, PLL (if selected), and SCI.
- Transfer these subroutines to RAM:
 - **LOADDATA** — Polls the SCI for data to be programmed, the start address, and length of data array (see [Message Structure to Communicate between Host and GP20](#)). This routine also calls the other RAM routines as needed.

- DUMPROW — Dumps the contents of the current row (64 bytes) out the SCI
- PRGFLSH — Controls the routine to program a row of FLASH
- ERACHK — Checks to see if FLASH needs to be erased and does so, if necessary
- DELNUS — Delays for n microseconds (n preloaded into H:X)
- PRGPAGE — Programs a page (eight bytes) of FLASH
- Execute code to program FLASH out of RAM, making necessary calls to other RAM routines to perform actual programming/erasing.
- Return to SCI port monitoring loop in RAM

RAM Utilization and Program Execution

Since program execution cannot occur out of a FLASH block at the same time it is being modified, and since this device has only one block of non-volatile memory, execution of code to modify the FLASH must be from RAM (or from monitor ROM which will be discussed later).

The program which resides in FLASH loads the necessary programming and erasing modules to RAM and then jumps to the main RAM routine which, when appropriate, makes calls to other RAM modules. Since RAM capacity in the GP20 is 512 bytes, the code executed in RAM is kept to that size.

Therefore, only the programming, erasing and dumping routines, the array of data to be programmed, necessary variables, and the stack fit into RAM. Almost all of RAM is used by the program, variables, message buffer, and stack.

The RAM utilization map is found in [Table 4](#).

Table 4. RAM Utilization Map

Function	Allocated Space	Address Range
Unused	16 bytes	\$40–\$4F
Download size	1 byte	\$50
Start address of block to program	2 bytes	\$51–\$52
Number of bytes to program	1 byte	\$53
Data array	64 bytes	\$54–\$93
Variables	20 bytes	\$94–\$A7
RAM routines	56 bytes	\$A8–\$DF
Stack	32 bytes	\$E0–\$FF
RAM routines	320 bytes	\$100–\$23F

Assembler Directives

A few switches in the program are implemented through assembler directives. They were left in the program for mode configuration and ease of user testing, especially in an emulation environment.

The directive constant and its use and meaning are outlined in [Table 5](#).

Table 5. Assembler Directives

Assembler Constant	Use
MONPROG	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.
TESTMOD	When set, changes all STA to LDA in programming/erase routines. Done to prevent an illegal write error when emulating. Programming is not attempted so any verification step will fail.
ERSDTST	When set, always causes erase verification to be true. Done to exercise good loop logic.

Other Application-Specific Memory and I/O Equates

Several other constants are used in the program which can be modified as desired by the user. This may be necessary if the user wishes to incorporate this program into an existing program which is already fixed in memory.

Table 6. Program Constants

Constant	Default Value	Other Possible Settings	Description
CPUSPD	2	4, 8	Specifies the bus frequencies: 2 = 2.45 MHz, 4 = 4.92 MHz, 8 = 8.0 MHz
RAM	\$50	\$40-\$4F	Specifies the start address of RAM. All RAM locations are offset from this value.
STCKSZ	\$23	Dependent on location of branch to page 2	Specifies the amount of RAM reserved for the stack
PRGSTRT	\$B000	Anywhere in FLASH	Start of the FLASH program that launches the RAM program
RAMPRG	\$A8	Anywhere after the variable space	Start of RAM program. Set here, just after the variable space
RAMPRSZ	\$1A7	Any value larger than the RAM program size plus the stack size, up to the end of RAM	Length of the RAM program plus the stack area
TESTDAT	\$C0	Any byte value	Value of all bytes, except size bytes, used in test mode instead of downloaded value. Set at this value so that the first address used (\$C0C0) is on a row boundary
PRGTRIES	20	Any value greater than 0	Number of attempts to program a page of FLASH before giving up
VTPGM VHLFTER VTKILL VTHVD VTHVTV VTVTP	1000 50,000 200 50 50 150	See programming timing specification in data book	Programming times
PLLCHK	0	1	Input port used to check if PLL is to be initiated; initiated when port is high
P, E, NHI, NLO, L, R	0, 1, 1, \$2C, \$80, 1	See PLL setup procedure in data book	PLL setup parameters; values used allow external clock of 32.768 kHz to be stepped up to 2.45-MHz bus frequency

Proper Clock Selection

A constant called CPUSPD is set in the application-specific memory and I/O (input/output) equates section of the program. Its purpose is to allow the programmer to select one of three bus frequencies for this constant, which directly affects how the SCI, the FLASH control register, and timing constants are set up.

In particular, based on a CPUSPD setting of 2, 4, or 8, corresponding to an internal frequency of 2.4576, 4.9152, or 8.0 MHz, the SCI is set to communicate at 9600 baud (assuming the selection of the internal bus clock as the SCI clock source (see [SCI Setup](#)) and the FDIV bit in the FLASH control register is set to cause the closest to the optimum setting of the charge pump frequency of 2 MHz.

If the external clock frequency is such that it causes a bus frequency lower than required for proper charge pump operation, the PLL can be enabled. The PLL is turned on upon initial program execution if port B, bit 0 is set. In this event, the program assumes an external clock of 32.768 kHz and steps this up to 2.4576-MHz bus frequency. An initialization routine sets the baud rate for the SCI to 9600 and the FDIV divider to divide-by-1 for a charge pump frequency of 2.4576 MHz.

Make sure that the CPUSPD constant described earlier is set to 2 if the PLL is enabled. Also, if an external clock frequency other than 32.768 kHz exists that would necessitate the use of the PLL (external clock less than 8 MHz), then the PLL setup and this SCI/FDIV initialization routine (SFINIT) will have to be modified for proper communication and charge pump frequency requirements. (See note in [FLASH Control Register](#).)

PLL Setup

Since this is not an application note on the use and control of the PLL, the description here is brief. The settings used are justified with calculations, and the means of making these settings are shown. The significance of each bit in each register is not covered. Refer to the clock generator module section in the GP20 data manual for details about proper setup.

The six control and status registers for the GP20's PLL are shown in [Figure 4](#).

Addr.	Register Name	Bit 7	6	5	4	3	2	1	Bit 0	
\$0036	PLL Control Register (PCTL)	Read:	PLLIE	PLLF	PLLON	BCS	PRE1	PRE0	VPR1	VPR0
		Write:								
		Reset:	0	0	1	0	0	0	0	0
\$0037	PLL Bandwidth Control Register (PBWC)	Read:	AUTO	LOCK	\overline{ACQ}	0	0	0	0	R
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0038	PLL Multiplier Select High Register (PMSH)	Read:	0	0	0	0	MUL11	MUL10	MUL9	MUL8
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0039	PLL Multiplier Select Low Register (PMSL)	Read:	MUL7	MUL6	MUL5	MUL4	MUL3	MUL2	MUL1	MUL0
		Write:								
		Reset:	0	1	0	0	0	0	0	0
\$003A	PLL VCO Select Range Register (PMRS)	Read:	VRS7	VRS6	VRS5	VRS4	VRS3	VRS2	VRS1	VRS0
		Write:								
		Reset:	0	1	0	0	0	0	0	0
\$003B	PLL Reference Divider Select Register (PMDS)	Read:	0	0	0	0	RDS3	RDS2	RDS1	RDS0
		Write:								
		Reset:	0	0	0	0	0	0	0	1

= Unimplemented
 R = Reserved

Notes:

1. When AUTO = 0, PLLIE is forced clear and is read-only.
2. When AUTO = 0, PLLF and LOCK read as clear.
3. When AUTO = 1, \overline{ACQ} is read-only.
4. When PLLON = 0 or VRS7:VRS0 = \$0, BCS is forced clear and is read-only.
5. When PLLON = 1, the PLL programming register is read-only.
6. When BCS = 1, PLLON is forced set and is read-only.

Figure 4. PLL Control and Status Registers

The intention is to boost the external clock of 32.768 kHz up to 2.45 MHz bus frequency. This is done through these steps:

- Set the desired VCO frequency to four times the desired bus frequency.

$$f_{VCLKDES} = 4 \times f_{BUSDES} = 4 \times 2.4576 \text{ MHz} = 9.83 \text{ MHz}$$

- Let the reference divider, R, which is represented by RDS3–RDS0 in the PCTL register, be set to 1 (default).
- Let the power-of-two multiplier, P, which is represented by PRE1–PRE0 in the PCTL register, be set to 0 (default) for this VCO frequency.
- Calculate the VCO frequency multiplier, N, which is represented by MUL11–MUL0 in the PMSH and PMSL registers, with this formula:

$$N = \text{round}[R \times f_{VCLKDES} / f_{RCLK}] = 300 \text{ (12 CH)}$$

where f_{RCLK} is the reference (external) frequency of 32.768 kHz

- Let the VCO power-of-two range multiplier, E, which is represented by VPR1–VPR0 in the PCTL register, be set to 1 for this VCO clock frequency.
- Calculate the center-of-range linear multiplier, L, which is represented by VRS7–VRS0 in the PMRS register, with this formula:

$$L = \text{round}[f_{VCLK} / (2^E \times f_{NOM})] = \text{round}[9.83 / (1 \times 38.4 \text{ kHz})] = 128 \text{ (80 H)}$$

where f_{NOM} is the range nominal multiplier for all operating voltage ranges

The previous settings will produce a VCO-programmed center-of-range frequency of:

$$f_{VRS} = L \times 2^E \times f_{NOM} = 128 \times 2 \times 38.4 \text{ kHz} = 9.83 \text{ MHz}$$

The only hardware requirements for using the PLL, aside from the oscillator circuit, are a filter capacitor on the CGMXFC pin, a bypass capacitor on the V_{DDA} pin, and a pullup resistor of about 10 K on PB0. Consult the data manual for recommended values for the capacitors.

SCI Setup

Registers

The SCI can be clocked from the external oscillator or from the internal bus clock. Selecting the clock is done through software by programming a bit, SCIBDSRC, in the CONFIG2 register. The default for this bit selects the external oscillator as the SCI clock source. If the external clock is very slow and is being stepped up by the PLL, as in this application, then it is necessary to use the internal bus as the clock source for the SCI to get a reasonably fast baud rate. This application does this to be able to derive 9600 bps transfer rate when the oscillator is at 32.768 kHz.

Figure 5 shows the structure and content of the CONFIG register. Description of the other two bits in this register will not be addressed here, other than to say that the PMPGVLVEN bit must be kept at its default state when operating with a V_{DD} above 3.6 volts.

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	0	0	0	0	0	PMPGVLVEN	OSC-STOPENB	SCIBD-SRC
Write:								
Reset:	0	0	0	0	0	0	0	0


 = Unimplemented

Figure 5. Configuration Register 2 (CONFIG2)

Several registers can be used to configure and monitor the SCI module. Most of the settings do not need to be changed from the default values for this application, so this discussion focuses on the values that need to be set or changed from the default.

The control and status registers for the SCI are shown in [Figure 6](#).

Addr.	Register Name	Bit 7	6	5	4	3	2	1	Bit 0	
\$0013	SCI Control Register 1 (SCC1)	Read:	LOOPS	ENSCI	TXINV	M	WAKE	ILTY	PEN	PTY
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0014	SCI Control Register 2 (SCC2)	Read:	SCTIE	TCIE	SCRIE	ILIE	TE	RE	RWU	SBK
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0015	SCI Control Register 3 (SCC3)	Read:	R8	T8	DMARE	DMATE	ORIE	NEIE	FEIE	PEIE
		Write:								
		Reset:	U	U	0	0	0	0	0	0
\$0016	SCI Status Register 1 (SCS1)	Read:	SCTE	TC	SCRF	IDLE	OR	NF	FE	PE
		Write:								
		Reset:	1	1	0	0	0	0	0	0
\$0017	SCI Status Register 2 (SCS2)	Read:							BKF	RPF
		Write:								
		Reset:	0	0	0	0	0	0	0	0
\$0018	SCI Data Register (SCDR)	Read:	R7	R6	R5	R4	R3	R2	R1	R0
		Write:	T7	T6	T5	T4	T3	T2	T1	T0
		Reset:	Unaffected by reset							
\$0019	SCI Baud Rate Register (SCBR)	Read:			SCP1	SCP0	R	SCR2	SCR1	SCR0
		Write:								
		Reset:	0	0	0	0	0	0	0	0

= Unimplemented R = Reserved U = Unaffected

Figure 6. SCI Control and Status Registers

This procedure is used in the program to set up the SCI:

- Enable the SCI by setting the ENSCI bit in SCC1.
- Enable both transmitting and receiving by setting bits TE and RE of SCC2.
- Set the baud rate for 9600 by writing to the baud rate register (SCBR) with a value that is dependent on the bus frequency, as shown in [Table 7](#).

Table 7. Baud Rate Register Settings

Bus Frequency	SCP1	SCP0	SCR2	SCR1	SCR0	Written to SCBR
8.0 MHz	1	1	0	0	0	\$30
4.9152 MHz	0	0	0	1	1	\$03
2.4576 MHz	0	0	0	1	0	\$02

All other settings are left as the defaults. Note that the program selects and programs the baud rate register based on the value of the CPUSPD constant as described in [Proper Clock Selection](#).

*On-Board Circuitry
Required
for RS-232
Communication*

To communicate to another device over an RS-232 line, voltage levels need to be adjusted so that the 0- or 5-volt signal transmitted or received by the GP20 controller gets converted to the ± 12 -volt signal used by the programming device and vice versa. This is implemented by a level translator IC (integrated circuit) and a few capacitors. A possible external circuit for this interconnection is shown in [Figure 7](#).

NOTE: *If another controller or other device is used which uses 0-volt and 5-volt signal levels for serial communication, then a level translator is not needed.*

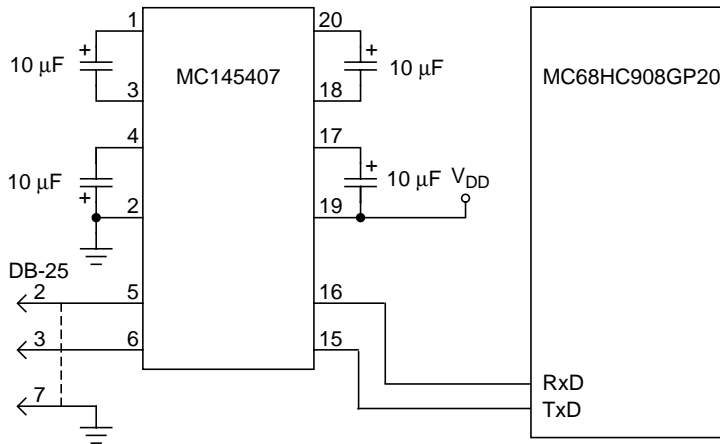


Figure 7. RS-232 Circuit for SCI Serial Communication

Message Structure to Communicate between Host and GP20

The structure of the data sent to the GP20 is simple, consisting of the components in [Table 8](#).

Table 8. Message Structure

Message Byte Location	Description	Final RAM Location
1	Count of the total number of bytes to be downloaded, including this byte	\$50
2–3	First address where the following data is to be programmed	\$51–\$52
4	Number of bytes to be programmed, or \$00 to just dump this referenced row, or a value between \$80 and \$FF to erase the entire FLASH array	\$53
5–68	Locations for 64 bytes of data to be programmed	\$54–\$93

Be aware that only the number of bytes specified in the first byte will be downloaded, so if byte 4, which specifies the number of bytes to be programmed is greater than the value in byte 1, erroneous data will be programmed. Also, if fewer than the number of bytes specified in byte 4 are included in the data block, erroneous data will be programmed.

NOTE: *If byte 4 is equal to 0, then nothing will be programmed. But the content of the row referenced by the first address bytes will be uploaded to the*

host. This first address, in this case, need not be the starting address for this row, but the 64 bytes downloaded will be within the row boundary. For example, if address \$B3B3 is downloaded with byte 4 being 0, then the 64 bytes in the range \$B380–\$B3BF will be downloaded. This may be useful when performing host verification without programming.

Note also that if byte 4 is a value between \$80 and \$FF, then the entire FLASH array will be erased. In this case, the first row of the erased FLASH (\$B000–\$B03F) is uploaded.

Uploads will not have header bytes. Instead, they will contain only 64 data bytes. The host message format should be sent with the same protocol for which the SCI has been initialized, namely 9600 baud, one start bit, one stop bit, and no parity.

Host Program

Any host program can be used to program a GP20 that is executing this FLASH programming program, provided that it conforms to the communication and message structure requirements specified in the [Message Structure to Communicate between Host and GP20](#). A Windows-based program has been developed for in-circuit programming of the GP20 specifically for this program and this application note. It can be downloaded at no cost from the Motorola web site <http://mot-sps.com/csic/techhelp/appsw/appsw.htm>.

CAUTION: *The installation and use of this host program is self-explanatory and, in fact, an explanation will not be offered. That is, since it is not a Motorola development tool product, no support is provided for the use of this program, and its use is at the user's risk.*

Programming the MC68HC908GP20 in Monitor Mode

General

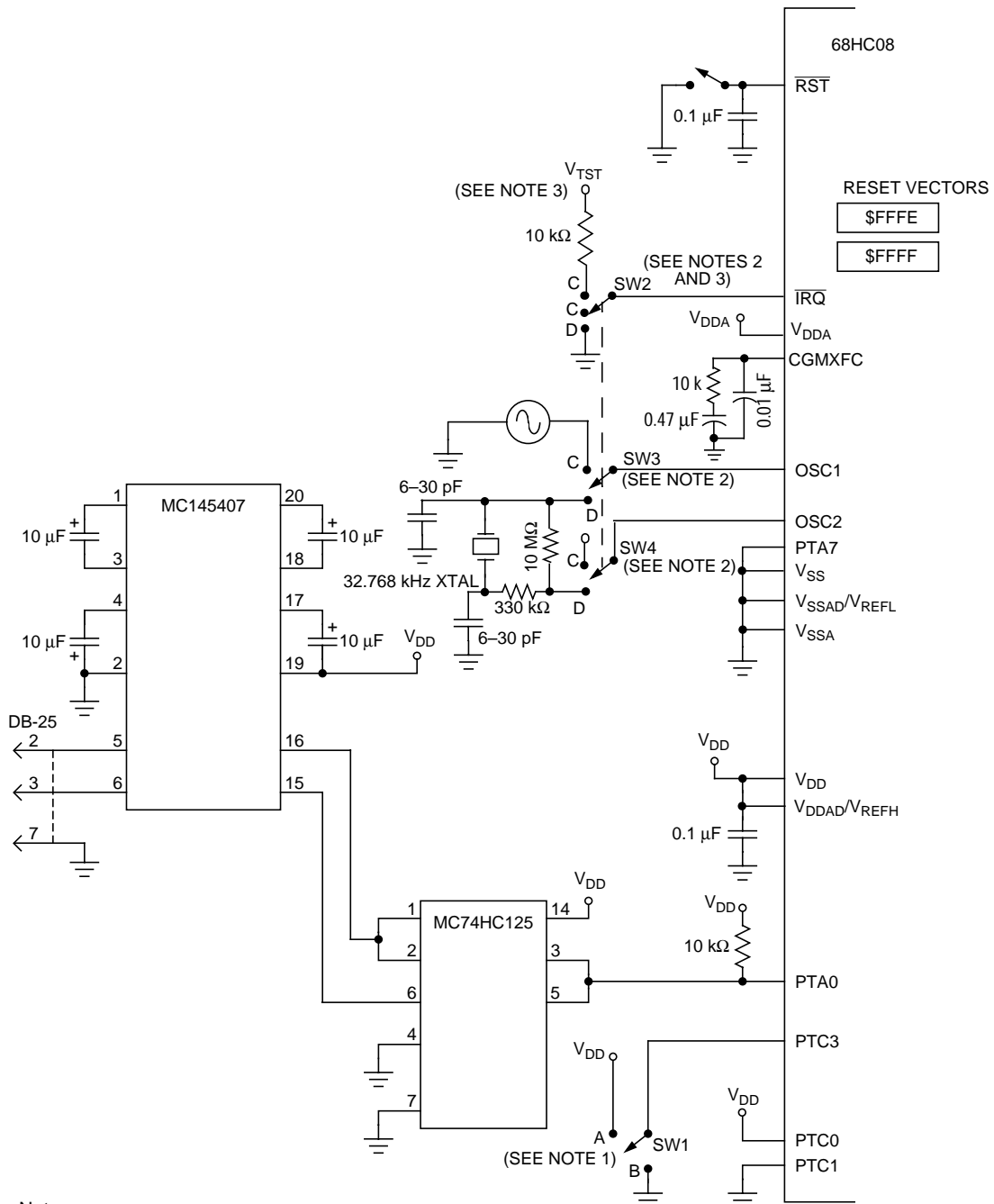
The MC68HC908GP20, like all other HC08 Family devices, contains a monitor utility which allows a host to control the device. Only a few very basic, low-level instructions are necessary and available to support this remote control. With the use of the six supported instructions, any location in the memory map can be read, any RAM location can be written to, or a small program can be loaded into and executed from the RAM area. One of the six instructions, READSP, allows for the reading of the stack pointer location, thereby giving the host control of where execution commences upon issuance of another instruction, RUN. Monitor mode is necessary for programming a blank device, since user mode programming requires an initial program in FLASH to load up the FLASH programming routines in RAM and jump to them for execution.

These monitor commands can be used to program the FLASH in the GP20. A couple of approaches can be adopted to implement monitor mode programming. One can use a host program to directly control the reading and writing of control registers, write to specific locations in FLASH, and cause the necessary delays during the FLASH programming cycle by only using the six monitor mode commands by a host program or user interface. The overhead here is the serial transmission time for each command. An alternate approach is to load a RAM program and then start its execution using the monitor mode commands. The RAM program monitors the same I/O (input/output) port, port A bit 0, for serial data/address transfers as that used when communicating with the monitor. Because of its inherent efficiency and its similarity to the user mode RAM program, this latter approach is what is used in this application note.

Serial communication between host and the GP20 in monitor mode is half-duplex where both transmission and reception of data is through a single bit port. The following sections discuss the circuit, communications, and security requirements to enter and use the GP20's monitor.

Circuit Requirements

The schematic in [Figure 8](#) shows the recommended circuit to enable entry into monitor mode for the 908GP20.



Notes:

- For monitor mode entry when $\overline{\text{IRQ}} = V_{\text{TST}}$:
SW1: Position A — Bus clock = $\text{CGMXCLK} \div 4$ or $\text{CGMVCLK} \div 4$
SW1: Position B — Bus clock = $\text{CGMXCLK} \div 2$
- SW2, SW3, and SW4: Position C — Enter monitor mode using external oscillator
SW2, SW3, and SW4: Position D — Enter monitor mode using external XTAL and internal PLL
- See [Table 9](#) for $\overline{\text{IRQ}}$ voltage level requirements.

Figure 8. Monitor Mode Circuit Requirements

Table 9 shows the requirements and options for entering monitor mode. In short, to enter monitor mode when the reset vector is not blank, PTC0 must be set high and PTC1 must be low. If the reset vector (\$FFFE–\$FFFF) is blank, then monitor mode can be entered without having high voltage ($V_{TST} = V_{DD} + 2.5 \text{ V}$) on the IRQ pin and any special configuration of port C pins. In this situation, the monitor checks to see if IRQ is low and if so, initializes the PLL for 2.4576-MHz bus frequency based on a 32.768-kHz oscillator frequency.

If the reset vectors are non-zero, then high voltage on IRQ is necessary and the automatic initialization of the PLL is not performed. This means that if the device is already programmed, then not only is test voltage required to get into monitor mode, but port C pins must be correctly configured and an external clock must be provided that can generate the desired internal bus frequency and baud rate without the use of the PLL. Of course, one could use the monitor commands to initiate the PLL, but this presupposes that a host can communicate with the device to set up the PLL. If an external clock of 32.768 kHz is used, the initial baud rate will be 64 bps.

Table 9. Monitor Mode Signal Requirements and Options

$\overline{\text{IRQ}}$	RESET	\$/FFFF/ \$/FFFF	PLL	PTC0	PTC1	PTC3	External Clock ⁽¹⁾	CGMOUT	Bus Frequency	COP	For Serial Communication			Comment
											PTA0	PTA7	Baud Rate ^{(2) (3)}	
X	GND	X	X	X	X	X	X	0	0	Disabled	X	X	0	No operation until reset goes high
V_{TST}	V_{DD} or V_{TST}	X	OFF	1	0	0	4.9152 MHz	4.9152 MHz	2.4576 MHz	Disabled	1	0	9600	PTC0 and PTC voltages only required if $\overline{\text{IRQ}} = V_{\text{TST}}$; PTC3 determines frequency divider
											X	1	DNA	
V_{TST}	V_{DD} or V_{TST}	X	OFF	1	0	1	9.8304 MHz	4.9152 MHz	2.4576 MHz	Disabled	1	0	9600	PTC0 and PTC1 voltages only required if $\overline{\text{IRQ}} = V_{\text{TST}}$; PTC3 determines frequency divider
											X	1	DNA	
V_{DD}	V_{DD}	\$/0000	OFF	X	X	X	9.8304 MHz	4.9152 MHz	2.4576 MHz	Disabled	1	0	9600	External frequency always divided by 4
											X	1	DNA	
GND	V_{DD}	\$/0000	ON	X	X	X	32.768 kHz	4.9152 MHz	2.4576 MHz	Disabled	1	0	9600	PLL enabled (BCS set) in monitor code
											X	1	DNA	
V_{DD} or GND	V_{TST}	\$/0000	OFF	X	X	X	X	—	—	Enabled	X	X	—	Enters user mode — will encounter an illegal address reset
V_{DD} or GND	V_{DD} or V_{TST}	Non-zero	OFF	X	X	X	X	—	—	Enabled	X	X	—	Enters user mode

Notes:

1. External clock is derived by a 32.768-kHz crystal or a 4.9152/9.8304-MHz off-chip oscillator.
2. PTA0 = 1 if serial communication; PTA0 = X if parallel communication
3. PTA7 = 0 → serial, PTA7 = 1 → parallel communication for security code entry
4. DNA = does not apply, X = don't care

Communication Protocol

The communication format which must be used when communicating with the GP20 in monitor mode is a non-return-to-zero mark/space format. The data format is one start bit, eight data bits, and one stop bit. Since the device probably will be set up to communicate at 9600 bps, the host should be set up to do the same.

Whatever the source of the reference clock, remember that the baud rate at which the device communicates is always the bus frequency divided by 256. This requires that if 9600 bps is desired, then the internal bus frequency must be 2.4576 MHz.

This frequency can be achieved by one of three means:

- Using an external oscillator at a frequency of 4.9152 MHz with PTC3 low
- Using an external oscillator at a frequency of 9.8304 MHz with PTC3 high
- Using an external oscillator at a frequency of 32.768 kHz with the IRQ pin low during reset to turn on the PLL; only offered when reset vector is low

NOTE: *The monitor mode serial interface uses a “bit banged” serial bit stream instead of the dedicated SCI protocol. This constrains the monitor mode baud rate, and, therefore, the bus frequency, to identically match the host baud rate. For example, a bus frequency of 2.5 MHz would yield a baud rate of 9766, which is within tolerance for the SCI but will not work with the monitor mode. This is the reason monitor mode frequencies are multiples of 2.4576 MHz.*

The monitor understands and processes six different commands.

Table 10 lists the commands, their functions, their opcode, and the total number of bytes required to send the command.

Table 10. Monitor Mode Command Set

Command	Function	Opcode	Number of Bytes Sent	Returned
READ	Read memory	\$4A	3	Value read; 1 byte
WRITE	Write to memory	\$49	4	None
IREAD	Indexed read of memory	\$1A	1	Two values read; 2 bytes
IWRITE	Indexed write to memory	\$19	2	None
READSP	Read stack pointer	\$0C	1	Address of stack pointer; 2 bytes
RUN	Run user program	\$28	1	None

It is important to note a couple of things regarding the host-to-monitor communication. They are:

- First, if using the circuit described in the prior section to connect to port A bit 0, then there will be a loop-back of the data from the host's transmit port to its receive port. This needs to be dealt with.
- Second, each command or data byte sent to the GP20 is echoed back to the host exactly one bit time after the stop bit for that byte is received. Therefore, adequate delay must be built into the host program to ensure that the next byte transmitted is sent at least one bit time after the echoed byte is received.

If at any time the received byte does not match what was sent, the command to abort execution of the last command can be sent by the host. This command is in the form of a break signal, which consists of 10 low bits including the start bit sent within 11 bit times of having received the echo of the last byte of the command to be aborted.

A complete command, with any follow-on address and/or data, must be sent before a break can be sent. Therefore, if an error is perceived after the echo of any byte of a command, the transmission of the entire command should be completed, followed by the transmission of the break signal. A full break signal will be echoed back to the host after a 2-bit time delay of having received one.

Also, note that the break signal sent by the host does not have to be 10 bits long and does not have to start exactly one bit time after a data

byte echo is received. All that is necessary is that at least one low bit, like the start bit of a transmitted byte, is sent within 1-bit time and within the 11-bit time period following the reception of the echo of the last byte of a command. At this point, the monitor code waits for the host to relinquish PTA0 by sensing a high signal after the low duration, and then it echoes a complete break signal to the host.

Refer to the GP20 data book, Motorola document order number HC908GP20GRS/D, for timing diagrams and further description of each of the six commands and break signal. Most of the commands are easily understood, but the sequence of operation to start execution of a program via the RUN command is worth discussing.

Follow this procedure to set up the stack and start execution of a loaded RAM program. This procedure assumes that the start of the RAM routine resides in the first page of RAM.

1. Issue the read stack pointer command (\$0C).
2. Monitor returns the high byte and then the low byte of the (SP+1) location. Ignore the high byte (it will be 0, since the stack pointer will be in the first page of memory) and add 4 to the low byte to determine the location to write the high byte of the RAM routine start address. Add 1 to this location value to determine the locations to write the low byte of the RAM routine start address.
3. Registers may be preloaded by writing their intended value to these locations:

X (MSB)	(SP+1)
CCR	(SP+1) + 1
ACC	(SP+1) + 2
X (LSB)	(SP+1) + 3

4. Write each of the values in step 3 to the appropriate locations using WRITE and IWRITE commands.
5. Issue the RUN command.

Example sequence:

To start execution of a RAM routine with a start location of \$A8, with all registers preloaded to \$00, and the condition code register preloaded with \$68:

1. Read stack pointer to get (SP+1) high byte
and (SP+1) low byte (\$0C)
2. Write \$00 to (SP+1) low byte (\$49, \$00, [(SP+1) low byte], \$00)
3. Indexed write \$68 to (SP+1) low byte+1 (\$19, \$68)
4. Indexed write \$00 to (SP+1) low byte+2 (\$19, \$00)
5. Indexed write \$00 to (SP+1) low byte+3 (\$19, \$00)
6. Indexed write \$00 to (SP+1) low byte+4 (\$19, \$00)
7. Indexed write \$A8 to (SP+1) low byte+5 (\$19, \$A8)
8. Issue RUN command. (\$28)

Security Requirements

The monitor has a security feature which requires the host to input a correct string of data before it can gain access to, and control of, the GP20. Without entering the exact sequence of data, which must match the data contained at locations \$FFF6–\$FFFD, access to the FLASH memory is disabled.

The 8-byte data stream may be entered any time after 256 cycles have elapsed following the rising edge of reset. The data and timing constraints must conform to the same protocol as stated in the previous section. The monitor will echo each byte input after a 1-bit time delay and the following byte entry must be at least two bit times after transmission of the echoed byte.

After all eight bytes have been received, the monitor sends out a break signal. If the received bytes match the eight bytes in FLASH, commands can then be sent for processing. If the received bytes do not match those in FLASH, monitor mode will continue, but access to the FLASH is

denied. In this case, any attempt to reference data in FLASH will return indeterminate data, and any attempt to execute from FLASH will result in an illegal address reset.

NOTE: *To try another security code sequence, remember to power down then power up the microcontroller before sending the new data string.*

Program Algorithm

The program included in this applications note is loaded into, and executed from, RAM. It can be used to fully or partially erase the FLASH and then reprogram the device while the device is running in monitor mode. This monitor mode RAM program is from the same program as the user mode RAM program described in [Programming the MC68HC908GP20 in User Mode](#). It is assembled with the MONMODE assembler directive set to signal that certain code specific to monitor mode programming be assembled.

The two modes of this program are similar in these respects:

- They both adhere to the same programming and erasing constraints as defined in [Program Algorithm](#).
- They both use the same six RAM modules, assembler directives, variables, and constants.
- RAM utilization is almost identical for the two programs.
- They both use the same message format as described in [Programming the MC68HC908GP20 in User Mode](#).

The primary differences are:

- Port A bit 0 is used for bidirectional communication in the monitor mode program instead of the SCI, for both RAM program loading as well as subsequent data downloading and device memory dumps/acknowledgments. The monitor mode program, of course, conforms to the communication protocol dictated by the monitor, rather than direct message passing used in the user mode program.

- The monitor mode program assumes an internal bus frequency of 2.45 MHz to generate the 9600 baud communication rate. If the bus frequency is different from this, then the baud rate will be proportionately different.
- The monitor mode program does not contain a module to set up the PLL, as the user mode program does, because of the built-in capability of the monitor to turn the PLL on.
- Instead of a load routine residing in FLASH and copying the necessary modules to RAM for execution, the monitor mode programs load the program into RAM from the host with the command set available in monitor mode. The program is first compiled and an S-record file is generated which the host program parses for downloading to the proper addresses in RAM.

Host Program

The host program for monitor mode programming used in this application note is the same as that used for user mode programming, that is, a Windows-based program executed on the PC. The only difference is that the host is put into an alternate mode in which it communicates with the target device in conformance with the monitor mode protocol requirements.

The basic functions of a suitable host program, including the one used in this application note, are:

- Allows entry of the eight security bytes and sends them to the GP20 in accordance with monitor mode timing requirements.
- Downloads the file (FP4.S19) containing the RAM program that the GP20 will utilize to receive serial data, program the FLASH, and dump its memory. This is done with a series of indexed write instructions.
- Initiates execution of the RAM program by configuring the data on the stack and then issuing the RUN command.

- Loads an S-record file with which to program the device. Alternatively, specifying the data to be programmed from within the program environment is also supported.
- Provides control logic to allow the operator to specify the range of FLASH to be programmed or erased and to communicate program data to the GP20 conforming to the message protocol that the RAM program expects. Additionally, the host should be able to provide verification of the data that is programmed and echoed back to it.

Conclusion

This application note describes a method of performing in-circuit serial programming of the FLASH memory in the MC68HC908GP20. The same general approach is followed for initial device programming, when monitor mode must be used, as when re-programming the device in user mode. In both situations, a program can be loaded into, and executed from, RAM which facilitates serial communication of data and commands from a host.

There are, of course, other ways of implementing FLASH programming.

One often-asked question is, "How many wires does it take to program the device?" Unfortunately, the answer to this question is not absolute, as it depends on how the user's target system is configured. In the best scenario, it takes only three wires to communicate with the target to program its FLASH.

This pre-supposes several existing or configurable conditions:

- The internal clock is adequate to generate an acceptable data rate to match the host's.
- The internal clock is high enough to create a charge pump frequency close to 2 MHz.
- Programming voltage can be generated on-board and can be applied to IRQ when necessary (when block protection is not set and when the reset vectors are not \$00).

- Communication data levels are at V_{DD} and ground potentials or RS-232 level translation is implemented on-board.
- For initial (blank-part) programming or reprogramming in monitor mode, monitor mode requirements can be met.

If any of these conditions cannot be met, then off-board circuitry, usually in the form of an interface board, is required and the number of connections to the target board increases.

This application note assumes that the above conditions can be met and that a PC is available which can serve as the host programmer. If this is the case, then nothing else is needed to do in-circuit serial programming.

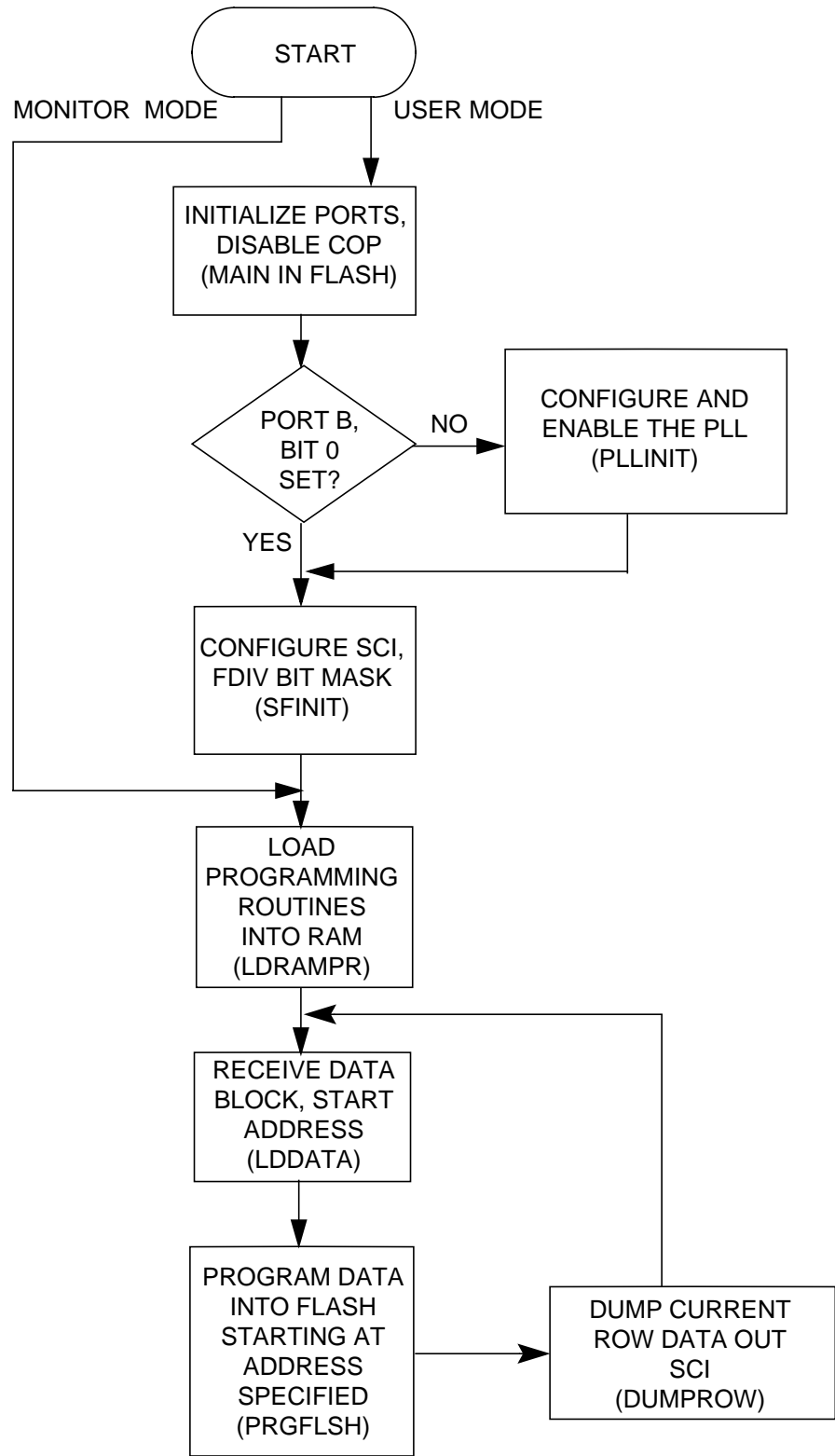


Figure 9. Main Program Flowchart

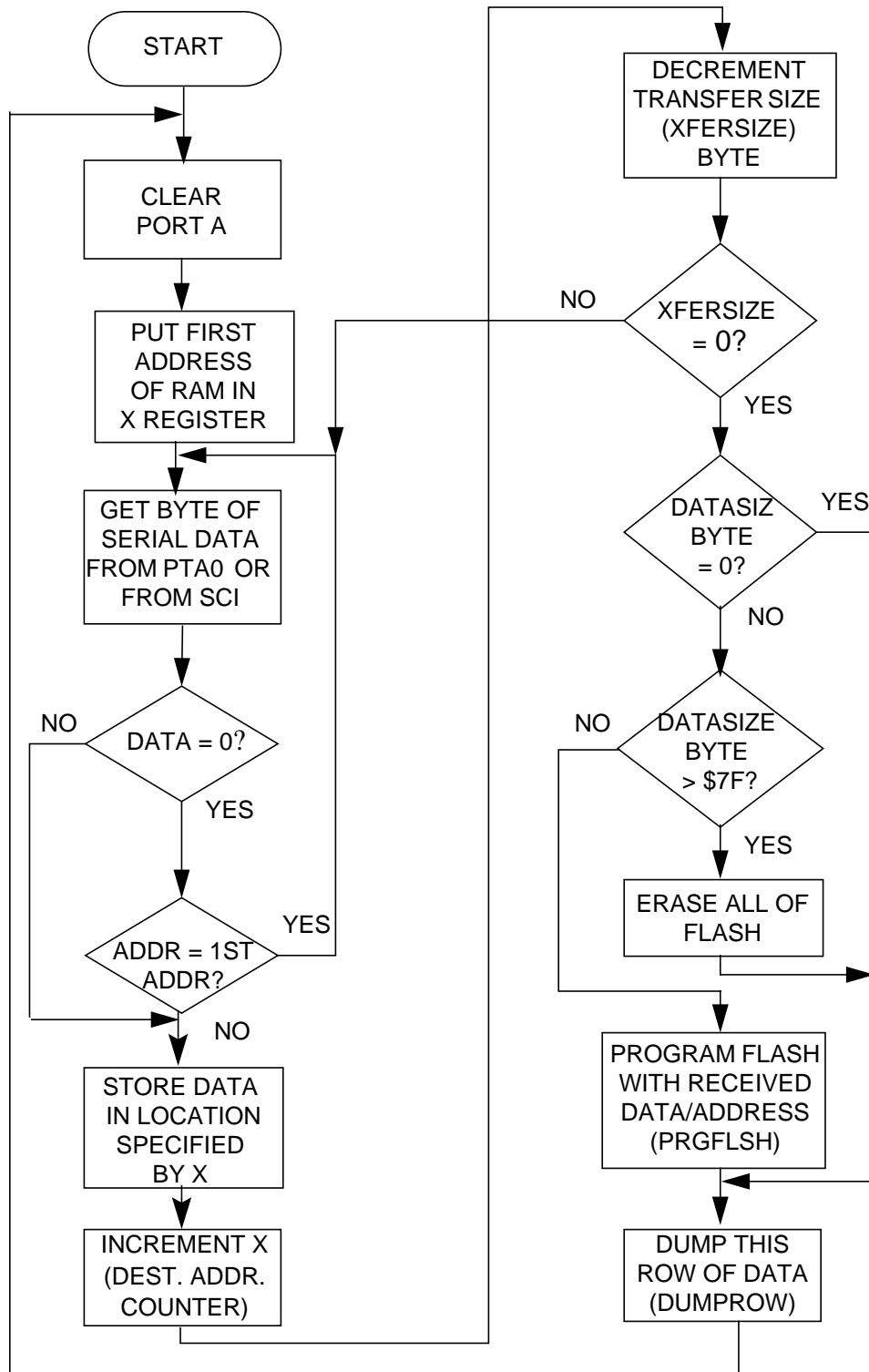


Figure 10. LDDATA Routine Flowchart

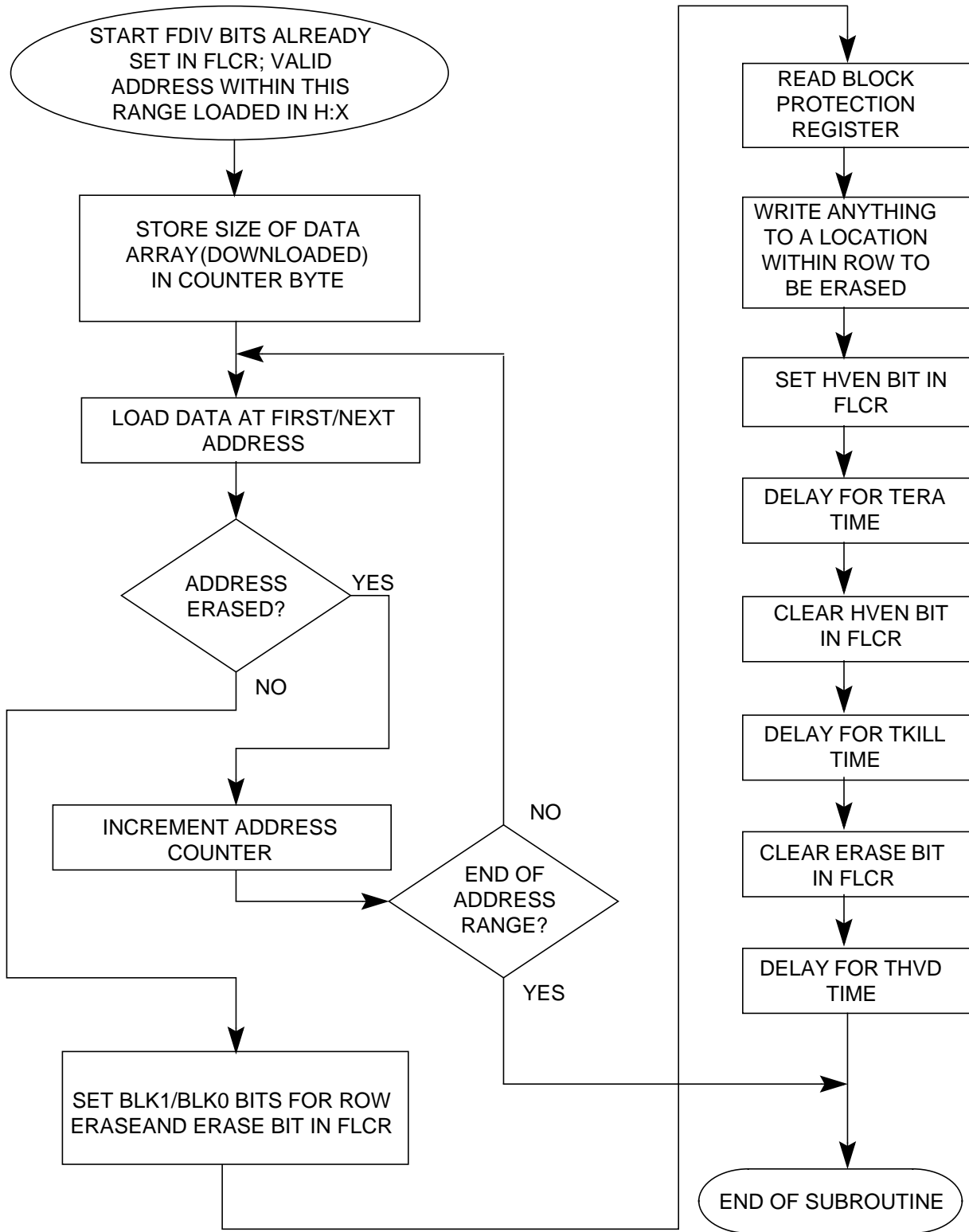


Figure 11. ERACHK Routine Flowchart

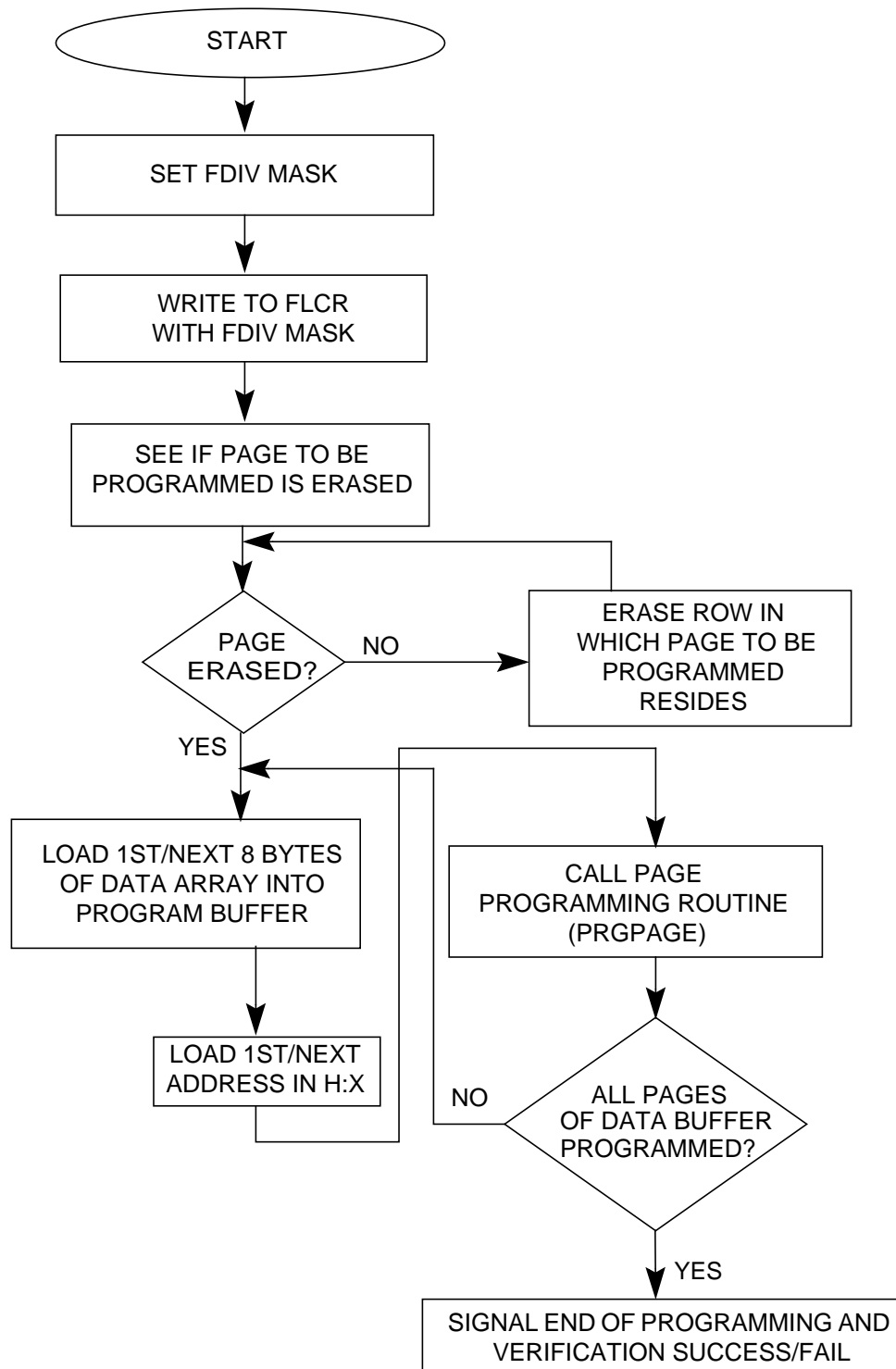


Figure 12. PRGFLSH Routine Flowchart

Source Code for FLASH Programming RAM Program

```
*****
* LISTING NO.: 1
*
* FILE NAME: FP4.ASM
* PURPOSE: To provide a FLASH erase, program and verify program
* TARGET DEVICE: HC908GP20
*
* MEMORY USAGE - RAM: 1A0H BYTES
*                 ROM: 280H BYTES
*
* ASSEMBLER: CASM08
* VERSION: 1.02
*
* PROGRAM DESCRIPTION:
* This program loads a RAM routine with instructions/data
* located in FLASH memory that:
*     Receives data over the SCI or Port A, bit 0
*     Row-erases FLASH block if necessary
*     Programs FLASH with received data
*     Dumps specified row out comm port
*     Bulk erases device upon command
*
* The program has assembler directives to be able to program in
* both user and monitor modes. In monitor mode, the generated S-record
* file will contain all of the necessary programming routines in RAM. 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      GRANT WHITACRE    03/04/98      INITIAL VERSION
* 0.1      GRANT WHITACRE    04/15/98      SECOND VERSION (FP2.ASM)-
*                                     LOADS ALL RTNS INTO RAM
*                                     SO RE-ENTRY INTO FLASH IS
*                                     NOT NECESSARY. ALLOWS
*                                     REPROGRAMMING OF ENTIRE
*                                     FLASH ARRAY.
```



```

P2          EQU      2
P1          EQU      1
P0          EQU      0
P7.         EQU      $80          ;BIT POSITION P7
P6.         EQU      $20
P4.         EQU      $10
P3.         EQU      $08
P2.         EQU      $04
P1.         EQU      $02
P0.         EQU      $01

***** DATA DIRECTION REGISTERS A-E *****
DDRA        EQU      $04          ;PORT A DATA DIRECTION REGISTER
DDRB        EQU      $05          ;PORT B DATA DIRECTION REGISTER
DDRC        EQU      $06          ;PORT C DATA DIRECTION REGISTER
DDRD        EQU      $07          ;PORT D DATA DIRECTION REGISTER
DDRE        EQU      $0C          ;PORT E DATA DIRECTION REGISTER

***** CONFIG 1 REGISTER *****
CONFIG1     EQU      $001F        ;CONFIG1 REGISTER
COPRS       EQU      7
LVISTOP     EQU      6
LVIRSTD     EQU      5
LVIPWRD     EQU      4
LVI5OR3     EQU      3
SSREC       EQU      2
STOP        EQU      1
COPD        EQU      0
COPRS.      EQU      $80
LVISTOP.    EQU      $40
LVIRSTD.    EQU      $20
LVIPWRD.    EQU      $10
LVI5OR3.    EQU      $08
SSREC.      EQU      $04
STOP.       EQU      $02
COPD.       EQU      $01

***** CONFIG 2 REGISTER *****
CONFIG2     EQU      $001E        ;CONFIG2 REGISTER
*           EQU      7
*           EQU      6
*           EQU      5
SEC         EQU      4          ;Security
*           EQU      3
*           EQU      2
OSCSTOPEN   EQU      1          ;Enable Oscillator during
                                ;STOP mode
SCIBDSRC    EQU      0          ;SCI baud rate clock source
* .         EQU      $80        ;BIT POSITION 7
* .         EQU      $40
* .         EQU      $20
SEC.        EQU      $10
* .         EQU      $08
* .         EQU      $04
OSCSTOPEN.  EQU      $02
SCIBDSRC.   EQU      $01

```

Application Note

```

***** FLASH CONTROL REGISTER *****
FLCR          EQU    $FE08          ;FLASH CONTROL REGISTER
FDIV1         EQU    7
FDIV0         EQU    6
BLK1          EQU    5
BLK0          EQU    4
HVEN          EQU    3
VERF          EQU    2          ;FOR 908GP20 ONLY
MARG          EQU    2          ;FOR 908XL36 ONLY
ERASE         EQU    1
PGM           EQU    0
FDIV1.        EQU    $80
FDIV0.        EQU    $40
BLK1.         EQU    $20
BLK0.         EQU    $10
HVEN.         EQU    $08
VERF.         EQU    $04
MARG.         EQU    $04
ERASE.        EQU    $02
PGM.          EQU    $01

***** BLOCK PROTECTION REGISTER *****
FLBPR         EQU    $FF80          ;FLASH BLOCK PROTECTION
                                           ;REGISTER
*             EQU    7
*             EQU    6
*             EQU    5
*             EQU    4
BPR3          EQU    3
BPR2          EQU    2
BPR1          EQU    1
BPR0          EQU    0
* .           EQU    $80
* .           EQU    $40
* .           EQU    $20
* .           EQU    $10
BPR3.         EQU    $08
BPR2.         EQU    $04
BPR1.         EQU    $02
BPR0.         EQU    $01

***** SCI REGISTERS *****
SCC1          EQU    $13          ;SCI CONTROL REGISTER 1
LOOPS         EQU    7          ;BIT #7
ENSCI         EQU    6
TXINV         EQU    5
M             EQU    4
WAKE         EQU    3
ILTY         EQU    2
PEN          EQU    1
PTY          EQU    0
LOOPS.        EQU    $80          ;BIT POSITION 7
ENSCI.        EQU    $40
TXINV.        EQU    $20
M.           EQU    $10
WAKE.         EQU    $08
ILTY.         EQU    $04

```

```

PEN.          EQU      $02
PTY.          EQU      $01

SCC2          EQU      $14          ;SCI CONTROL REGISTER 2
TIE           EQU      7           ;Transmit interrupt enable
TCIE          EQU      6
RIE           EQU      5
ILIE          EQU      4           ;Idle line interrupt enable
TE            EQU      3           ;Transmit enable
RE            EQU      2           ;Receive enable
RWU           EQU      1           ;Receiver wakeup enable
SBK           EQU      0           ;Send break
TIE.          EQU      $80
TCIE.         EQU      $40
RIE.          EQU      $20
ILIE.         EQU      $10
TE.           EQU      $08
RE.           EQU      $04
RWU.          EQU      $02
SBK.          EQU      $01

SCC3          EQU      $15          ;SCI CONTROL REGISTER 3
R8            EQU      7           ;Bit 8 receive (for 9-bit characters)
T8            EQU      6           ;Bit 8 transmit
DMARE         EQU      5
DMATE         EQU      4
ORIE          EQU      3
NEIE          EQU      2
FEIE          EQU      1
PEIE          EQU      0
R8.           EQU      $80
T8.           EQU      $40
DMARE.        EQU      $20
DMATE.        EQU      $10
ORIE.         EQU      $08
NEIE.         EQU      $04
FEIE.         EQU      $02
PEIE.         EQU      $01

SCS1          EQU      $16          ;SCI STATUS REGISTER 1
SCTE          EQU      7           ;BIT #7
TC            EQU      6
SCRF          EQU      5
IDLE          EQU      4
OR            EQU      3
NF            EQU      2
FE            EQU      1
PE            EQU      0
SCTE.         EQU      $80          ;BIT POSITION 7
TC.           EQU      $40
SCRF.         EQU      $20
IDLE.         EQU      $10
OR.           EQU      $08
NF.           EQU      $04
FE.           EQU      $02
PE.           EQU      $01

```

Application Note

```

SCS2      EQU      $17      ;SCI STATUS REGISTER 2
*         EQU      7        ;BIT #7
*         EQU      6
*         EQU      5
*         EQU      4
*         EQU      3
*         EQU      2
BKF       EQU      1
RPF       EQU      0
*.        EQU      $80      ;BIT POSITION 7
*.        EQU      $40
*.        EQU      $20
*.        EQU      $10
*.        EQU      $08
*.        EQU      $04
BKF.      EQU      $02
RPF.      EQU      $01

SCDR      EQU      $18      ;SCI Data
RDR       EQU      $18      ;SCI Receive Data (same as SCDR)
TDR       EQU      $18      ;SCI Transmit Data (same as SCDR)

SCBR      EQU      $19      ;SCI BAUD RATE REGISTER
*         EQU      7
*         EQU      6
SCP1      EQU      5        ;SCI prescaler sel bit 1 00=clk/1,
                           ;01=clk/3
SCP0      EQU      4        ;SCI prescaler sel bit 0 10=clk/4,
                           ;11=clk/13
RES       EQU      3
SCR2      EQU      2        ;SCI baud rate sel bit 2
                           ;000=/1...111=/128
SCR1      EQU      1        ;SCI baud rate sel, bit 1
SCR0      EQU      0        ;SCI baud rate sel, bit 0
*.        EQU      $80
*.        EQU      $40
SCP1.     EQU      $20
SCP0.     EQU      $10
RES.      EQU      $08
SCR2.     EQU      $04
SCR1.     EQU      $02
SCR0.     EQU      $01

```

* APPLICATION-SPECIFIC MEMORY AND I/O EQUATES

* THE VALUE FOR CPUSPD DRIVES THE FDIV SETTING AND THE BAUD RATE

* PRESCALER FOR THE SCI.

* MAKE SURE THAT CPUSPD IS SET TO 2 IF THE PLL IS TO BE USED.

```

CPUSPD    EQU      2        ;2 = 2.45 MHZ OPER. FREQ.
                           ;4 = 4.92 MHZ, 8 = 8.0 MHZ

```

* ABS. ADDRESS OF MONITOR ROUTINES COMMENTED OUT IN TEST VERSION

```

GET_PUT   EQU      $FE97    ;MON RTN TO GET A BYTE ON
                           ;PTA0 AND ECHO BACK

```

Application Note
Source Code for FLASH Programming RAM Program

```

PUT_BYTE      EQU      $FEAA      ;SEND A BYTE OUT PTA0
MONRTNS      EQU      $B300      ;MONITOR ROUTINES INCLUDED IN
;PROGRAM FOR TESTING
RSTVLOC      EQU      $FFFE      ;RESET VECTOR LOCATION

RAM          EQU      $50        ;FIRST ADDRESS OF RAM, ACTUALLY
; $40 BUT WILL START AT $50

NXTPAGE      EQU      $100
STCKSIZ      EQU      $18
PRGSTRT      EQU      $F000      ;START OF FLASH PROGRAM
LASTBYT      EQU      $F0F0
NXFPAGE      EQU      $F100
RAMPRG       EQU      RAM+$58    ;START OF RAM ROUTINE

RAMPRSZ      EQU      $197      ;320 BYTES (MAX)

TESTDAT      EQU      $C0
XFRCODE      EQU      PRGSTRT+RAMPRG
STUBINT      EQU      PRGSTRT+$240 ;PUT HERE FOR NOW -
;SHOULD BE SAFE
VECSTRT      EQU      $FFDC      ;START OF USER VECTOR AREA

PRGTRIES     EQU      25        ;MAX. NUMBER OF PROGRAM TRIES

* MASKS FOR ERASE RANGE
FARMASK      EQU      $00        ;FULL ARRAY
HARMASK      EQU      $10        ;HALF ARRAY
RW8MASK      EQU      $20        ;8 ROWS
ROWMASK      EQU      $30        ;1 ROW

* PROGRAMMING TIMES - CHANGE THESE VALUES IF NECESSARY TO
* CHANGE TIMES! ALL TIMES ARE IN MICROSECONDS
VTPGM        EQU      1000      ;1000 - PROGRAM TIME
VHLFTER      EQU      50000     ;1/2 ERASE TIME
VTKILL       EQU      200       ;HIGH-VOLTAGE KILL TIME
VTHVD        EQU      50        ;RETURN TO READ TIME
VTHVTV       EQU      50        ;HVEN LOW TO VERF HIGH TIME
VTVTP        EQU      150       ;VERF HIGH TO PGM LOW TIME

* INTERMEDIATE PROGRAMMING TIMES (CALCULATION PURPOSES ONLY)
CTPGM        EQU      (VTPGM/6) ;DIVIDE BY 6 HERE TO NORMALIZE
CHLFTER      EQU      (VHLFTER/6) ;FOR NEXT STEP. PADDED FROM
CTKILL       EQU      (VTKILL/6) ;8 TO 6 TO COMPENSATE FOR ODD
CTHVD        EQU      (VTHVD/6) ;FREQUENCIES (2.45 AND 4.92 MHZ)
CTHVTV       EQU      (VTHVTV/6) ;AND TRUNCATION.
CTVTP        EQU      (VTVTP/6)

* CPU SPEED-CORRECTED PROGRAMMING TIMES (TIMES ACTUALLY USED)
TPGM         EQU      (CPUSPD*CTPGM) ;Program time = 1000 µs @ 8 MHz
HLFTERA      EQU      (CPUSPD*CHLFTER) ;Half of Erase time = 50 µs @ 8 MHz
TKILL        EQU      (CPUSPD*CTKILL) ;HV Kill time = 200 µs
THVD         EQU      (CPUSPD*CTHVD) ;Return to read time = 50 µs
THVTV        EQU      (CPUSPD*CTHVTV) ;HVEN low to VERF high time = 50 µs
TVTP         EQU      (CPUSPD*CTVTP) ;VERF high to PGM low time = 150 µs

```

Application Note

```

* PLL EQUATES
PLLCHK      EQU      0                ;PLL CHECK BIT ON PORT B

* 908GP20 PLL REGISTERS
PCTL        EQU      $0036            ;PLL Control Register
PBWC        EQU      $0037            ;PLL Bandwidth Control Register
PMSH        EQU      $0038            ;PLL Multiplier Select Register High
PMSL        EQU      $0039            ;PLL Multiplier Select Register Low
PVRS        EQU      $003A            ;PLL VCO Range Select Register
PRDS        EQU      $003B            ;PLL Reference Divider Select Register

* Initial Settings for 32.768 kHz crystal clock to produce a 2.4576 MHz
*internal clock
P           EQU      0                ;PLL Prescaler Program Bits (PRE)
;value of PCTL (def = 0)
E           EQU      1                ;PLL VCO Power-of-Two Range Select Bits
;(VCR) value of PCTL (def = 0)
NHI         EQU      1                ;PLL Multiplier Select Bits (MUL)
;value of PMSH (def = 0)
NLO         EQU      $2C              ;PLL Multiplier Select Bits (MUL)
;value of PMSL (def = 0)
L           EQU      $80              ;PLL VCO Range Select Bits (VRS)
R           EQU      0                ;PLL Reference Divider Select Bits
;(RDS) value of PRDS (def=1)
;0 value for R or N is
;interpreted as a 1
AUTO        EQU      7                ;Bit 7 of PBWC
LOCK        EQU      6                ;Bit 6 of PBWC
PLLON       EQU      5                ;Bit 6 of PCTL
BCS         EQU      4                ;Bit 4 of PCTL

```

```

*****
* VARIABLE DEFINITIONS & RAM SPACE USAGE
*****

```

```

* $40-$4F    NOT USED                ( 16 BYTES)
* $50        TRANSFER SIZE            ( 1 BYTE)
* $51-$52    FIRST ADDRESS TO BE PROGRAMMED( 2 BYTES)
* $53        DATA SIZE (DATASIZ)    ( 1 BYTE)
* $54-$93    DATA ARRAY              ( 64 BYTES)
* $94-$A7    VARIABLES                ( 20 BYTES)
* $A8-$EF    RAM PROGRAM              ( 72 BYTES)
* $F0-$FF    STACK                    ( 16 BYTES)
* $100-$23F  RAM PROGRAM              (320 BYTES)
* $50-$23F   TOTAL                    (512 BYTES)

```

```

ORG      RAM
XFRSIZE  RMB      1                ;NUMBER OF BYTES TO BE TRANSFERRED
FRSTADR  RMB      2                ;FIRST ADDR TO BE PROGRAMMED
DATASIZ  RMB      1                ;NUMBER OF BYTES TO PROGRAM
DATARAY  RMB      64               ;RESERVE 64 BYTES FOR DATA
DATA1    RMB      8                ;DATA TO BE PROGR. IN PAGE-8 BYTES
REPROG   RMB      1                ;A $01 HERE SIGNALS NEED TO REPROGRAM
TEMPH    RMB      1                ;RAM COUNTER (HI BYTE)
TEMPL    RMB      1                ;RAM COUNTER (LOW BYTE)
TEMP2    RMB      1                ;TEMP. HOLDING LOC. FOR TRANSFERS/PR
;TRIES

```



```

FDIVMSK      RMB      1          ;FDIV OF FLCR BIT MASK
BYTECNT      RMB      1          ;BYTE COUNT USED DURING PAGE PROG.
ARAYIDX      RMB      1          ;INDEX INTO THE DATA ARRAY
CURRBYT      RMB      1          ;CURRENT BYTE BEING READ/WRITTEN (0-7)
STATBYT      RMB      1          ;STATUS OF PROGRAM SUCCESS -
                                     ;BIT 7 = FAILED; BIT 6 = SUCCEDED
*****
*      Program Algorithm (User Mode Programming)
*      1.      Initialize all variables and ports, PLL (if selected) and
*              SCI.
*      2.      Monitor SCI 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
*              C.      ERABLK
*              D.      DELNUS
*              E.      PRGPAGE
*      4.      Jump to first byte of main RAM program (RAMPRG).
*      5      Execute RAM program MAINPRG and then return to SCI
*              port monitoring loop in RAM.
*
*      Program Algorithm - Monitor Mode Programming
*      1.      Monitor PTA0 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
*              monitoring loop in RAM.
*****
*      START OF PROGRAM
*****
IFNE MONPROG
    ORG      PRGSTRT
START      EQU      *
            CLR      PORTA
            CLR      PORTB
            MOV      #$02,DDR0          ;USING PTB1 AS output FOR PLL INIT.
            MOV      #$31,CONFIG1      ;DISABLE THE COP AND LVI
            BRSET    PLLCHK,PORTB,STFDV ; (IF PB0 = 1 THEN PLL OFF)
            JSR      PLLINIT
STFDV     JSR      SFINIT              ;SET FDIV BITS ACCORDING TO CPU SPEED
                                                ;ALSO INITIALIZES THE SCI
            JSR      LDRAMPR          ;LOAD ENTIRE RAM PROGRAM

*      If testing code in FLASH to retain labels, then take following jump...
*      JMP      XFRCODE              ;LOAD DATA INTO RAM FROM SCI
*      ...otherwise jump to RAM to execute
*      JMP      RAMPRG
*****

```

Application Note

```
* NAME: PLLINIT
* PURPOSE: INITIALIZES THE PLL
* ENTRY CONDITIONS: NONE
* EXIT CONDITIONS: NONE
* SUBROUTINES CALLED:
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF FLASH
* THE FOLLOWING INITIALIZES THE PLL FOR 2.4576 MHZ INTERNAL CLOCK
* BASED ON 32.768 KHZ EXTERNAL CRYSTAL CLOCK SO WE CAN CREATE A
* STANDARD BAUD RATE (9600) FOR COMMUNICATION AND AN ACCEPTABLE
* CHARGE PUMP FREQUENCY.
*****
PLLINIT:
* Instruction Setup for 32.768 kHz => 2.4576 MHz (31 bytes total)
* P = 0, R = 1, E = 1, NHI = 1, NLO = 2CH, PVRS(L) = 80H
      CLR      PCTL                ;turn PLL off (on by default)
      BSET    0,PCTL              ;set E=1 (VPR0=1)
      BSET    0,PMSH              ;N (hi byte) = 1
      MOV     #NLO,PMSL          ;N (lo byte) = predefined
      MOV     #L,PVRS            ;bit 6 on by default, L=80H
      BSET    PLLON,PCTL
      BSET    AUTO,PBWC          ;put in auto bandwidth mode
      BRCLR   LOCK,PBWC,*
      BSET    BCS,PCTL

* TEST CODE FOR PLL SETUP
* Following tests the above PLL settings to see if the internal
* clock is set at the desired rate. Internal clock rate is 16x
* frequency sensed at bit 1 of port A.
*TESTPLL   BSET    1,DDRB        ;bit 1 set as output
*BITOFF    BCLR    1,PORTB      ;4 cycles
*
*          NOP
*
*          NOP                    ;3 cycles
*BITON     BSET    1,PORTB      ;4 cycles
*          BRCLR   0,PORTB,BITOFF ;5 CYCLES
*
*          RTS                    ;16 cycles

*****
* NAME: SFINIT
* PURPOSE: INITIALIZES THE SCI FOR DATA RECEPTION;
*          INITIALIZES THE FDIV BITS
* ENTRY CONDITIONS:
* EXIT CONDITIONS:
* SUBROUTINES CALLED:
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF FLASH
* DATA SHOULD BE IN THE FORM OF 1 START-BIT, 8 DATA-BITS,
* 1 STOP-BIT. BAUD RATE IS SET FOR 9600, BASED ON CPUSPD
*****
SFINIT
      BSET    SCIBDSRC,CONFIG2
      LDA     #CPUSPD            ;SET FDIV MASK FOR CURRENT CPU SPEED
      NSA
      LSLA
```

Application Note
Source Code for FLASH Programming RAM Program

```

        BCC      NOT8MH
        MOV      #$30,SCBR          ;fX/64/9600 = 13 (SCBR=$30)
        MOV      #$C0,FDIVMSK
        BRA      XSCINIT
NOT8MH  LSLA
        BCC      NOT4MH
        MOV      #$03,SCBR          ;fX/64/9600 = 8 (SCBR=$03)
        MOV      #$80,FDIVMSK
        BRA      XSCINIT
NOT4MH  MOV      #$02,SCBR          ;fB/64/9600 = 4 (SCBR=$02)
        CLR      FDIVMSK
XSCINIT
        BSET     ENSCI,SCC1          ;TURN THE SCI ON
        MOV      #$0C,SCC2          ;SET SCCR2 INITIAL VALUE
                                           ;TURNS ON TE & RE
        RTS
*****
*****
* 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+,TEMP2         ;TRANSFER LOCATION IN RAM
        PSHH
        PSHX
        LDHX      TEMPH            ;LOAD ADDRESSES THAT HOLD THE DEST.
        MOV       TEMP2,X+         ;TRANSFER DATA FROM TRANSFER LOCATION
NEXT     STHX      TEMPH
        CPHX      #RAMPRG+RAMPRSZ ;TO NEXT LOCATION AT RAM DESTINATION
        PULX
        PULH
        BEQ       XLDRAMP
        CPHX      #LASTBYT         ;SEE IF RAM DESTINATION IS LAST BYTE
        BNE      NXTMOVE          ;IN PAGE 1 AND IF SO, INCREMENT THE
        LDHX      #NXFPAGE
        PSHH
        PSHX
        LDHX      TEMPH
        LDHX      #NXTPAGE
        STHX      TEMPH
        PULX
        PULH
        BRA      NXTMOVE          ;IF NOT DONE, CONTINUE
XLDRAMP RTS
ENDIF

```

Application Note

```

* START OF CODE TO BE TRANSFERRED TO RAM
IFNE MONPROG
    ORG        XFRCODE                ;CURRENTLY $B0A8
    ENDIF

* OR - START OF THE MONITOR PROGRAM WHICH WE'LL ORG IN RAM
IFEQ MONPROG
    ORG        RAMPRG                 ;CURRENTLY $A8
START:
    ENDIF
*****
* NAME: LDDATA
* PURPOSE: LOAD RAM WITH USER'S DATA AND START ADDRESS VIA THE SCI;
*          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    $40
*                 BYTES TO BE SENT (INCL. THAT BYTE)
*   2-3            THE FIRST ADDRESS WHERE THE     $41-$42
*                 FOLLOWING DATA IS TO BE PROGRAMMED
*   4              NUMBER OF BYTES TO BE PROGRAMMED $43
*   5-68           ARRAY SPACE FOR DATA TO BE PROGRAMMED $44-$83
*
* 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 SCI. MUST RESET
* AFTER THE LAST ROW DOWNLOAD.
* IF A DATA ARRAY IS RECEIVED WITH A NUMBER OF BYTES TO BE PROGRAMMED OF >= $80
* 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.
*****
LDDATA        CLRH
              CLRA
              LDX        #RAM                ;POINT TO START OF RAM
IFEQ MONPROG
              CLR        FDIVMSK
ENDIF

WAITRX:
IFEQ MONPROG
              JSR        GET_PUT
              NOP
              NOP
ENDIF
IFNE MONPROG
              BRCLR     SCRF,SCS1,*          ;WAIT FOR RX REGISTER TO FILL
              LDA       SCS1                ;PART 1 OF CLEARING THE SCRF BIT
              LDA       SCDR                ;PART 2 OF CLEARING THE SCRF BIT
              ;READ DATA BYTE FROM RX REGISTER
ENDIF

```

```

CHKCHK      CPX      #RAM          ;IF VALUE OF 1ST BYTE IS ZERO, THEN
            BNE      STORNOW       ;BAD START - KEEP LOOPING FOR NON-
            TSTA
            BEQ      WAITRX        ;ZERO FIRST BYTE
STORNOW     STA      ,X            ;STORE THE DATA IN RAM
            INCX     ;MOVE TO NEXT RAM LOCATION
            DBNZ    RAM,WAITRX     ;DEC. PROG SIZE CNTR (1st BYTE)
            ;IF ENTIRE PROG NOT LODED, CONT.
            LDA      DATASIZ       ;IF SIZE OF DATA TO BE PROGRAMMED
            BEQ      DUMP          ;IS 0, THEN ONLY DUMP THIS ROW.
            BPL     JUSTPRG
            CLRA
            ORA      #FARMASK
            BSR     JERASE         ;ERASEIT
            BRA     DUMP
JUSTPRG     BSR     PRGFLSH
DUMP        BSR     DUMPROW
            BRA     LDDATA

```

```

*****
* NAME: DUMPROW
* PURPOSE: DUMPS THE ENTIRE 64-BYTE ROW THAT THE START ADDR IS IN
* ENTRY CONDITIONS: H-X CONTAINS THE NEXT ADDR TO BE PROGRAMMED
* EXIT CONDITIONS: NONE
* SUBROUTINES CALLED: ERACHK
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF RAM
*****

```

```

DUMPROW     LDHX    FRSTADR        ;PUT FIRST ADDR IN H-X
            TXA
            AND     #$C0          ;PUT ON ROW BOUNDARY
            TAX
RDBYTE      LDA     ,X
WAITTX
IFEQ MONPROG
            JSR     PUT_BYTE      ;SEND OUT PTA0
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
ENDIF
IFNE MONPROG
            BRCLR  SCTL,SCS1,WAITTX ;WAIT FOR TRANSMIT REG. TO EMPTY
            PSHA
            LDA     SCS1
            PULA
            STA     SCDR          ;SEND EEPROM DATA TO SERIAL OUTPUT
ENDIF
            INCX     ;MOVE TO NEXT ADDRESS
            TXA
            AND     #$3F
            BNE    RDBYTE        ;IF NOT FINISHED, CONTINUE
ENDDUMP     RTS

```

Application Note

```
*****
* FOLLOWING IS RELOCATABLE, DEPENDING ON WHERE THE PAGE BREAK FOR THE
* STACK IS TO BE LOCATED. MUST PLACE IT SO THAT THE BRANCH TO THE NEXT
* PAGE IS AT LEAST PAST $FF.
JERASE      BSR      ERASEIT      ;NEED A 1/2 BRANCH HERE
LASTBYTE    RTS
            ORG      LASTBYTE+STCKSIZ ;STCKSIZ NEEDS TO BE SET AS EQUATE
*****
```

```
*****
* NAME: PRGFLSH
* PURPOSE: ERASES (IF NECESSARY), PROGRAMS DATA IN DATA ARRAY
*          AT LOCATION SPECIFIED, AND THEN VERIFIES.
*          DATA MUST BE WITHIN A ROW BOUNDARY.
* ENTRY CONDITIONS: NONE
* EXIT CONDITIONS: NONE
* SUBROUTINES CALLED: JERACH (ERACHK), JPRGPAG (PRGPAGE)
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF FLASH
*              PROGRAMMING ALGORITHM
* 1. LOAD FIRST ADDRESS; CLEAR BYTCNTR
* 2. SEE IF ADDRESS IS ON PAGE BOUNDARY. IF NOT,
*   LOAD EXISTING DATA FROM CORRECT LOCATIONS IN FLASH
*   TO FILL IN THE BEGINNING OF THE PAGE.
* 3. LOAD DATA FROM DATA BUFFER TO FILL (REST OF) PAGE OR
*   UNTIL THE END OF THE DATA ARRAY IS REACHED, WHICHEVER
*   COMES FIRST.
* 4. IF DATASIZ WAS NOT BIG ENOUGH TO COMPLETE CURRENT PAGE,
*   THEN LOAD EXISTING DATA FROM CORRECT LOCATIONS IN
*   FLASH TO FILL IN THE END OF THE PAGE.
* 5. INCREMENT BYTECNT ACCORDING TO HOW MANY BYTES OF DATA
*   WERE USED FOR CURRENT PAGE.
* 6. PROGRAM PAGE WITH THE EIGHT BYTES OF LOADED DATA.
* 7. PERFORM VERIFICATION CHECK TO SEE IF PAGE WAS PROGRAMMED.
*   IF NOT, THEN SEND A HIGH OUT THE "VERIFICATION FAILED" PORT
*   AND RETURN TO FLASH.
* 8. CHECK TO SEE IF BYTECNT = DATASIZ. IF SO THEN RETURN
*   TO FLASH. IF NOT, GO TO 3.
* THIS ROUTINE CHECKS TO SEE IF THE FIRST ADDRESS OF THE DATA
* TO BE PROGRAMMED IS ON A PAGE BOUNDARY, AND THE LAST ADDRESS
* OF DATA TO BE PROGRAMMED IN AT THE END OF A PAGE. SINCE WE PROGRAM
* WHOLE PAGES AT A TIME, WE'LL JUST REPROGRAM THOSE BYTES IN THE
* PAGE BEFORE THE FIRST ADDRESS (IF NECESSARY) AND THOSE BYTES IN
* THE PAGE AFTER THE LAST ADDRESS (IF NECESSARY) WITH THE VALUE
* THAT ALREADY EXISTS THERE IN FLASH.
*****
```

```
PRGFLSH
        BSET      1,PORTB
        CLRA
        ORA      FDIVMSK
        STA      FLCR
        BSR      ERACHK
        CLR      BYTECNT
        LDHX     FRSTADR
        PSHX
        PSHH
```

```

NOSTUFF
* FOLLOWING LOADS REST OF OR ALL 8 BYTES INTO THE DATA BUFFER
      CLR      ARAYIDX
NXTLOAD  CLRH
      LDX      ARAYIDX
      CPX      DATASIZ          ;SEE IF ALL DATA IN
                                ;ARRAY HAS BEEN LOADED
      BEQ      NOMODAT
      LDA      DATARAY,X       ;WHERE X CONTAINS INDEX INTO ARRAY
      INC      ARAYIDX
      LDX      BYTECNT
      STA      DATA1,X
      INCX
      STX      BYTECNT
      CPX      #08
      BNE      NXTLOAD
      PULH
      PULX
      BSR      JPRGPAG
      PSHX
      PSHH
      CLR      BYTECNT
      BRA      NXTLOAD
NOMODAT  PULH
      PULX
      RTS

*****
* NAME: ERACHK
* PURPOSE: CHECKS TO SEE IF RANGE TO BE PROGRAMMED NEEDS TO BE
* ERASED FIRST, AND ERASES IF NECESSARY.  ERASE BITS (BLK0, BLK1)
* AND FDIV BITS ALREADY SET IN FPCR; VALID ADDRESS FOR THIS RANGE
* LOADED IN H:X.  THIS ROUTINE DOES NOT VERIFY ERASE, AND THEN DO A NUMBER
* OF ATTEMPTS TO RE-ERASE.  MAYBE LATER IF ENOUGH ROOM.
* ENTRY CONDITIONS: NONE
* EXIT CONDITIONS: NONE
* SUBROUTINES CALLED: DELNUS
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF RAM
* IF THE RANGE TO BE PROGRAMMED IS NOT ALREADY ERASED, THIS
* ROUTINE WILL AUTOMATICALLY ERASE THE ROW THAT THIS DATA IS IN.
*****
ERACHK  MOV      DATASIZ,BYTECNT          ;WANT TO KEEP THIS VALUE
                                ;USED AS A COUNTER HERE
      LDHX      FRSTADR          ;LOAD THE FIRST ADDR IN H:X
NXTCHK  LDA      ,X              ;LOAD THE DATA AT THIS ADDR
IFEQ  ERSDTST
      CLRA
ENDIF
      BNE      ERAROW          ;IF NOT ZERO, THEN ERASE ROW
      AIX      #01
      DBNZ     BYTECNT,NXTCHK
      BRA      XERACHK
ERAROW  CLRA
      ORA      #ROWMASK
ERASEIT  LDHX     FRSTADR          ;RELOAD ADDRESS IN CASE ENTRY WAS AT ERASEIT

```

Application Note

```

        ORA      FDIVMSK
        ORA      #ERASE.
        STA      FLCR          ;SET FDIV MASK BASED ON CPU SPEED
ERABLK  LDA      FLBPR
IFEQ TESTMOD
        LDA      ,X
ENDIF
IFNE TESTMOD
        STA      ,X
ENDIF
        LDHX    #FLCR          ;FOLLOWING SETS THE HVEN BIT IN FLCR
        LDA      ,X
        ORA      #HVEN.
        STA      ,X
        LDHX    #HLFTERA      ;DELAY FOR 1/2 OF TERA
        BSR     DELNUS
        LDHX    #HLFTERA      ;DELAY FOR 1/2 OF TERA
        BSR     DELNUS
        LDHX    #FLCR          ;CLEARS THE HVEN BIT
        LDA      ,X
        EOR     #HVEN.
        STA      ,X
        LDHX    #TKILL        ;DELAY FOR TKILL
        BSR     DELNUS
        LDHX    #FLCR          ;CLEAR ERASE BIT
        LDA      ,X
        EOR     #ERASE.
        STA      ,X
        LDHX    #THVD         ;DELAY FOR THVD
        BSR     DELNUS
XERACHK RTS
JPRGPAG BSR     PRGPAGE      ;NEEDED TO STAY IN RANGE
        RTS

```

```

*****
* NAME: DELNUS
* PURPOSE: DELAY N  $\mu$ s
* ENTRY CONDITIONS: H-X CONTAINS THE TIME DELAY (IN  $\mu$ s.)
* EXIT CONDITIONS: PRESERVES THE CONTENTS OF ACC
* SUBROUTINES CALLED: NONE
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF RAM
*****

```

```

DELNUS   PSHA
        PSHH
        PULA
* FOLLOWING LOOP EXECUTES H-X NUMBER OF TIMES; 99.6% OF THE
* TIME THIS LOOP IS 8 CYCLES IN DURATION WHICH IS 1  $\mu$ S @ 8 MHz
D1US     TSTX          ;(1) X
        BNE     NOADEC      ;(3) X
        TSTA          ;(1)
        BEQ     XDELNUS     ;(3)
        DECA          ;(1)
NOADEC   DECX          ;(1) X
        BRA     D1US        ;(3) X
XDELNUS  PULA
        RTS                ;(4) RETURN AFTER WANTED DELAY

```



```

*****
* NAME: PRGPAGE
* PURPOSE: PROGRAMS A PAGE (8 BYTES) OF FLASH
* ENTRY CONDITIONS: H-X REG. LOADED WITH FIRST ADDRESS TO BE
* PROGRAMMED; DATA1-DATA8 HAS BEEN LOADED
* EXIT CONDITIONS: NONE
* SUBROUTINES CALLED: DELNUS
* EXTERNAL VARIABLES USED:
* DESCRIPTION: EXECUTED OUT OF RAM
*****
PRGPAGE      PSHA                      ;(A) SAVE CONTENTS OF ACCUMULATOR
             MOV      #PRGTRIES,TEMP2
PRGLOOP      CLR      REPROG
             LDA      FDIVMSK          ;SET FDIV MASK FOR CURRENT CPU SPEED
             ORA      #PGM.           ;SET PGM BIT
             STA      FLCR            ;WRITE THIS TO THE FLASH CONTROL REG.
             LDA      FLBPR          ;READ FROM BLOCK PROT. REG.
             PSHH                     ;(B)
             PSHX                     ;(C) PUSH LO BYTE OF ADDR TO STACK
             CLR      BYTECNT        ;SET BYTE COUNT TO 0
             CLRX
LDNOTHR      CLRH
             LDA      DATA1,X
             PULX                     ;(C') POP LO BYTE OF ADDR BACK INTO X
             PULH                     ;(B')
IFEQ TESTMOD
             LDA      ,X              ;READ INSTEAD OF WRITE DURING TESTING
                                     ;TO PREVENT ILLEGAL ADDRESS ACCESS
ENDIF
IFNE TESTMOD
             STA      ,X              ;STORE DATA TO ADDR SPEC. BY H-X
ENDIF
             AIX      #$01            ;INCREMENT THE ADDRESS
             PSHH                     ;(B) PUSH LO BYTE OF ADDR BACK TO STACK
             PSHX                     ;(C)
             INC      BYTECNT        ;INCREMENT THE BYTE COUNTER
             LDX      BYTECNT        ;LOAD X WITH BYTE COUNT
             CPX      #$08
             BNE     LDNOTHR
             PULX                     ;C'
             PULH                     ;B'
             AIX      #-1
             PSHH                     ;B
             PSHX                     ;C
             LDHX     #FLCR          ;FOLLOWING SETS THE HVEN BIT IN FLCR
             LDA      ,X
             ORA      #HVEN.
             STA      ,X
             PSHX                     ;(D) PUSH FLCR (LO BYTE) TO STACK
             PSHH                     ;(E) PUSH FLCR (HI BYTE) TO STACK
             LDHX     #TPGM
             BSR      DELNUS
                                     ;FOLLOWING CLEARS THE HVEN BIT
             PULH                     ;(E') POP FLCR (HI BYTE) FROM STACK
             PULX                     ;(D') POP FLCR (LO BYTE) FROM STACK
             LDA      ,X
             EOR      #HVEN.

```

Application Note

```

        STA      ,X
        PSHX                    ;(D) PUSH FLCR (LO BYTE) TO STACK
        PSHH                    ;(E) PUSH FLCR (HI BYTE) TO STACK

        LDHX      #THVTV        ;DELAY FOR THVTV
        BSR      DELNUS

                                ;SET THE MARG BIT
        PULH                    ;(E') POP FLCR (HI BYTE) FROM STACK
        PULX                    ;(D') POP FLCR (LO BYTE) FROM STACK
        LDA      ,X
        ORA      #MARG.

        PSHX                    ;(D) PUSH FLCR (LO BYTE) TO STACK
        PSHH                    ;(E) PUSH FLCR (HI BYTE) TO STACK
        LDHX      #TVTP        ;DELAY FOR TVTP
        BSR      DELNUS
        BRA      CLRPGM

HALFBRA  BRA      PRGLOOP

CLRPGM   PULH                    ;CLEAR THE PGM BIT
                                ;(E') POP FLCR (HI BYTE) FROM STACK
                                ;(D') POP FLCR (LO BYTE) FROM STACK
        PULX
        LDA      ,X
        EOR      #PGM.
        STA      ,X
        LDHX      #THVD        ;DELAY FOR THVD
        BSR      DELNUS

* NOW READ WHAT'S BEEN PROGRAMMED AND CHECK IT WITH DATA1-DATA8
* BYTECNT IS ALREADY SET AT 8 SO LEAVE IT
        PULX                    ;(C') POP PG LST ADDR (LO-B) FRM STACK
        PULH                    ;(B') POP PG LST ADDR (HI-B) FRM STACK
        MOV      #DATA1+7,CURRBYT
RDNOTHR  LDA      ,X            ;NOW READ DATA AND STORE THEM
        DECX                    ;DEC THIS ADDR NOW BEFORE PUSHING IT
        PSHX                    ;(B) PUSH PG ADDR (LO-B) TO STACK
        PSHH                    ;(C)
        CLRH
        LDX      CURRBYT
        CMP      ,X
        PULH                    ;(C')
        PULX                    ;(B') POP PG ADDR (LO-B) FR STACK
        BNE      FAILVER
        DEC      CURRBYT
        DBNZ    BYTECNT,RDNOTHR ;DECREMENT THE BYTE COUNTER
        BRA      PASSVER

FAILVER  MOV      #$01,REPROG   ;STORE A VALUE IN REPROG TO SIGNAL A
                                ;REPROG OF PAGE

PASSVER  INCX

        TXA                    ;PUT ADDR OF 1ST BYTE OF THIS PAGE
        AND      #$F8          ;INTO H:X REGARDLESS OF FAIL/PASS
        TAX
        PSHH                    ;(B) PUSH PG 1ST ADDR (LO-B) TO STACK
        PSHX                    ;(C)
        LDHX      #FLCR        ;FOLLOWING CLEARS THE MARG BIT IN FLCR
        LDA      ,X
        EOR      #MARG.
        LDA      REPROG

```

```


PTPA          BEQ      PASSED
              DEC      TEMP2
              PULX
              PULH          ;(C') POP PG 1ST ADDR (LO-B) FR STACK
              BNE      HALFBRA      ;(B') POP PG 1ST ADDR (HI-B) FR STACK
              BRA      FAILJMP      ;TRY IT AGAIN

PASSED        PULX          ;(C')
              PULH          ;(B')
FAILJMP       AIX      #08      ;INDEX TO NEXT PAGE
XPRGPAG       PULA
ENDRAMP        RTS          ;(A') RESTORE ACCUM

IFNE MONPROG
*****
* INTERRUPT SERVICE ROUTINES
*****
              ORG      STUBINT
CLRINT        RTI
*****
* INTERRUPT AND RESET VECTORS
*****
              ORG      RSTVLOC
RSTVEC        FDB      START
*****
ENDIF          ;PUT HERE TO TAKE OUT VECTOR
              ;IN MONPROG S19 FILE

```

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, 1-800-441-2447 or 1-303-675-2140. Customer Focus Center, 1-800-521-6274

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 141, 4-32-1 Nishi-Gotanda, Shinagawa-ku, Tokyo, Japan. 03-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

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/>

Mfax is a trademark of Motorola, Inc.



MOTOROLA

© Motorola, Inc., 1999

AN1770/D