Using MC68HC908 On-Chip
FLASH Programming Routines

by Kazue Kikuchi
MCD Applications Engineering
Austin, Texas

1 Introduction

In this application note, the following devices are supported:

MC68HC908EY16  MC68HRC908JL3
MC68HC908GR4    MC68HLC908JL3
MC68HC908GR4A   MC68HC908JL3E
MC68HC908GR8    MC68HRC908JL3E
MC68HC908GR8A   MC68HLC908JL3E
MC68HC908GT16   MC68HC908KX2
MC68HC908J8     MC68HC908XX8
MC68HC908JK1    MC68HC908QT1
MC68HRC908JK1   MC68HC908QT2
MC68HLC908JK1   MC68HC908QT4
MC68HC908JK1E   MC68HC908QY1
MC68HRC908JK1E  MC68HC908QY2
MC68HLC908JK1E  MC68HC908QY4
MC68HC908JK3    MC68HLC908QT1
MC68HRC908JK3   MC68HLC908QT2
MC68HLC908JK3   MC68HLC908QT4
MC68HC908JK3E   MC68HLC908QY1
MC68HRC908JK3E  MC68HLC908QY2
MC68HLC908JK3E  MC68HLC908QY4
MC68HC908JL3

Table of Contents

1 Introduction ............................................. 1
2 Page Erase Issue ....................................... 2
3 The Routines ........................................... 2
  3.1 GETBYTE ........................................... 3
  3.2 RDVRNRNG .......................................... 4
  3.3 PRGRNGE .......................................... 4
  3.4 ERARNGE — Page Erase ............................. 5
  3.5 ERARNGE — Mass Erase ............................ 6
  3.6 DELNUS ........................................... 6
4 Device-Specific Information Related to
  On-chip FLASH Routines ............................... 8
5 Variables ................................................. 8
6 The Data Structure ..................................... 9
7 Addresses of Routines ................................. 10
8 MC68HC908JX8 Trim Routine ........................... 10
9 Typical Routine Calls ................................... 10
  9.1 Example for GETBYTE ............................... 11
  9.2 Examples for RDVRNRNG ......................... 11
  9.3 Example for Page Erase Operation ............... 12
  9.4 Examples for Mass Erase Operation ............. 12
  9.5 Examples for PRGRNGE ......................... 13
  9.6 Example for DELNUS .............................. 14
10 Page Erase Workaround ............................... 14
11 ROM Routines Source Code ............................ 15
12 Workaround Code ..................................... 29

© Freescale Semiconductor, Inc., 2005. All rights reserved.
The original purpose of this application note was to describe how to use the on-chip FLASH routines residing in ROM (read-only memory). In the first version of the note, only a few devices were supported.

This updated application note not only supports the original purpose but also accomplishes the following purposes:

1. There is a bug on the page erase operation using the on-chip FLASH erase routine described in the original note. In this note, the bug is described and its workaround is provided. For details, refer to Section 2, “Page Erase Issue.”
2. More devices that have the on-chip FLASH routines are supported.
3. Information is updated to provide additional useful information to the user.

## 2 Page Erase Issue

The page erase issue does not apply to all devices supported in this note. The issue applies to the following devices:

- MC68HC908GR4
- MC68HC908GR4A
- MC68HC908GR8
- MC68HC908GR8A
- MC68HC908JB8
- MC68HC908JK1
- MC68HRC908JK1
- MC68HLC908JK1
- MC68HRC908JK1E
- MC68HC908JK3
- MC68HRC908JK3
- MC68HLC908JK3
- MC68HC908JK3E
- MC68HRC908JK3E
- MC68HLC908JK3E
- MC68HC908JL3
- MC68HRC908JL3
- MC68HLC908JL3
- MC68HC908JL3E
- MC68HRC908JL3E
- MC68HLC908JL3E
- MC68HC908KX2
- MC68HC908KX8

If the user target device is not one of the above, skip this section. The device does not have the page erase issue.

One of the on-chip FLASH routines, named ERARNGE, supports FLASH erase and was meant to erase a page size of FLASH or the whole FLASH array.

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

The issue is caused by servicing the computer operating properly (COP) during the t\text{Erase} delay routine. The workarounds for this issue are:

1. Write your own page erase routine and keep it in FLASH. Whenever a page erase is desired, copy the routine to RAM and execute the erase from there. Be sure to omit instructions that service the COP.
2. Refer to Section 10, “Page Erase Workaround.” In the workaround, ERARNGE located in ROM is used so that the user does not need to develop his/her own routine. The new page erase routine is called in a similar manner to the original. However, additional RAM is required to perform this routine.

## 3 The Routines

The collection consists of five callable routines, which are described in Table 2. These routines are explained briefly here, but the parameters and the passing method are addressed in later sections.
3.1 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. Table 3 shows which communication port (COMMPORT) is used for each device.

GETBYTE expects the same non-return-to-zero (NRZ) communication protocol and baud rate that is used in monitor mode. 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.

This routine detects a framing error when a STOP bit is not detected. If the carry (C) bit of the condition code register (CCR) is cleared after returning from this routine, a framing error occurred during the data receiving process. Therefore, the data in the accumulator is not reliable. The user software is responsible for handling such errors.

The communication baud rate is defined by the internal operating bus frequency ($f_{op}$) divided by a constant value. Table 1 shows the divider value and a typical baud rate for each device.

<table>
<thead>
<tr>
<th>Divider Value</th>
<th>Typical Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>9600bps @ $f_{op}$=2.4576MHz</td>
</tr>
<tr>
<td>256</td>
<td>9600bps @ $f_{op}$=2.4576MHz</td>
</tr>
<tr>
<td>256</td>
<td>9600bps @ $f_{op}$=2.4576MHz</td>
</tr>
<tr>
<td>306</td>
<td>9600bps @ $f_{op}$=3MHz</td>
</tr>
<tr>
<td>256</td>
<td>9600bps @ $f_{op}$=2.4576MHz</td>
</tr>
<tr>
<td>256</td>
<td>9600bps @ $f_{op}$=2.4576MHz</td>
</tr>
<tr>
<td>256</td>
<td>9600bps @ $f_{op}$=2.4576MHz when ECGST bit in OSCSTAT is set</td>
</tr>
<tr>
<td>208</td>
<td>4800bps @ $f_{op}$=1MHz when ECGST bit in OSCSTAT is cleared</td>
</tr>
</tbody>
</table>
The Routines

For the MC68HLC908QT/QY.xx devices, the divider value depends on the status of the external clock status (ECGST) bit in the oscillator status (OSCSTAT) register. When this bit is set, the divider value of 256 is selected. And when this bit is cleared, the divider value of 208 is selected.

To use GETBYTE, the COMMPORT pin must have an external pull-up.

Interrupts are not masked (the I bit is not set) and the COP is not serviced in this routine. User software should ensure that interrupts are blocked during character reception.

3.2 RDVRRNG

RDVRRNG routine serves two purposes:

- **Send-Out Option** — Used to read a range of FLASH locations and to send the read data through COMMPORT.
- **Verify Option** — Used to read a range of FLASH locations and to verify the read data against DATA array.

3.2.1 Send-Out Option

If the accumulator is initialized with $00 when entering RDVRRNG, then the data read will be sent to the monitor mode communication port (COMMPORT). Therefore, the communication baud rate is the same as the baud rate described in Section 3.1, “GETBYTE.” When this option is selected, the COMMPORT pin must be pulled up and configured as an input and the communication data bit must be initialized to 0.

3.2.2 Verify Option

If the accumulator is initialized with a non-zero value, the read data is verified against the DATA array for each byte of FLASH, and the DATA array is replaced by the data read from FLASH. The carry (C) bit of the condition code register (CCR) is set if the data in the specified range is verified successfully against the data in the DATA array.

Both options calculate a checksum on data read in the range. 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.

To select a range, the beginning and end of the FLASH range to be read are specified as parameters to FADDR and LADDR. These parameters are explained in more detail later in Section 5, “Variables.”

Interrupts are not masked. COP is serviced in RDVRRNG. However, the COP timeout might still occur in the send-out option if COP is configured for a short timeout period.

3.3 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 parameters called FADDR and LADDR, respectively. 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. This routine can be
used in conjunction with RDVRRNG to perform a complete program and verification cycle of the specified range.

PRGRNGE allows any range to be passed to it. That is, the range does not have to be coincident with row boundaries\(^1\). 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 user must ensure only two things:

- The range specified is first erased
- The data for the specified range must be in the data array in RAM

Because this routine calls the delay routine DELNUS to generate proper delays, parameter CPUSPD must be set correctly when calling PRGRNGE. Parameters required for this routine are explained in more detail later in Section 5, “Variables.”

Interrupts are masked (the I bit is set) and COP is serviced in this routine.

**NOTE**

Regarding the JB8 and JL/JKxx(E), the FLASH block protect register (FLBPR) does not consist of FLASH. Since FLASH is always protected after reset, unprotect the locations to be programmed in the user software before calling this routine.

### 3.4 ERARNGE — Page Erase

The page erase operation using the ERARNGE routine supports the following devices:

<table>
<thead>
<tr>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC68HC908EY16</td>
</tr>
<tr>
<td>MC68HC908GT16</td>
</tr>
<tr>
<td>MC68HC908QT1</td>
</tr>
<tr>
<td>MC68HC908QT2</td>
</tr>
<tr>
<td>MC68HC908QT4</td>
</tr>
<tr>
<td>MC68HC908QTY1</td>
</tr>
<tr>
<td>MC68HC908QTY2</td>
</tr>
<tr>
<td>MC68HC908QTY4</td>
</tr>
<tr>
<td>MC68HLC908QT1</td>
</tr>
<tr>
<td>MC68HLC908QT2</td>
</tr>
<tr>
<td>MC68HLC908QT4</td>
</tr>
<tr>
<td>MC68HLC908QTY1</td>
</tr>
<tr>
<td>MC68HLC908QTY2</td>
</tr>
<tr>
<td>MC68HLC908QTY4</td>
</tr>
</tbody>
</table>

If the user target device is not in the above list, refer to Section 2, “Page Erase Issue.”

ERARNGE can be called to erase a page of 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 within a page to be erased. To select the page erase operation, a control variable in RAM called CTRLBYT is used. Writing $00 to CTRLBYT selects the page erase operation. To set proper delays, CPUSPD must be set correctly. CPUSPD and CTRLBYT are explained in more detail later in Section 5, “Variables.”

Interrupts are masked. In the EY16 and GT16 ERARNGE, COP is not serviced. On the other hand, in the QT/QYxx ERARNGE, COP is serviced. Servicing COP does not cause the page erase issue for these devices.

Regarding the MC68HLC908QT/QYxx devices, only 1 MHz operating frequency (\(f_{op}\)) is supported so that CPUSPD must be set with a value $04.

---

1. The flexible boundary condition does not apply for the MC68HLC908QT/QYxx devices. When FLASH is programmed using this routine for these devices, all bytes that will be programmed must be in the same row. Furthermore, only a 1 MHz operating frequency (\(f_{op}\)) is supported so that CPUSPD must be set with a value $04. COP is not serviced.
NOTE
Regarding the EY16 and GT16, $t_{\text{Erase}}$ delay is set with 1ms in the page erase operation using this routine. Therefore, this setup supports less than 1000 program/erase cycles. On the other hand, the QT/QYxx ERARNGE uses $t_{\text{Erase}}$ delay 4ms in this routine. This setup supports more than 1000 program/erase cycles.

3.5 ERARNGE — Mass Erase

All devices listed in this note can use ERARNGE to execute the mass erase operation.

ERARNGE can be called to erase the entire FLASH. This routine does not use the last address (LADDR) variable. The first address (FADDR) placed in H:X can be any address within the FLASH. To select a mass erase operation, a control variable in RAM called CTRLBYT is used. Writing $40$ to CTRLBYT selects the mass erase operation. To set the proper delay, CPUSPD must be set correctly. CPUSPD and CTRLBYT are explained in more detail later in Section 5, “Variables.”

Regarding the MC68HLC908QT/QYxx devices, only 1 MHz operating frequency ($f_{\text{op}}$) is supported so that CPUSPD must be set with a value $04$.

Interrupts are masked. In the EY16 and GT16 ERARNGE, COP is not serviced. In the QT/QYxx ERARNGE, COP is serviced.

NOTE
Regarding the JB8 and JL/JKxx(E), the FLASH block protect register (FLBPR) does not consist of FLASH. Since FLASH is always protected after reset, unprotect the whole FLASH array in the user software before calling this routine.

3.6 DELNUS

DELNUS is a delay routine used in support of PRGRNGE and ERARNGE. It can, however, be called independently in the user software. DELNUS uses two parameters stored in the accumulator (A) and the X register (X). Neither of these parameters is passed as an absolute value. The total delay (cycles) resulting from this routine is:

\[
\text{DELNUS} = 3 \times (A \text{ value}) \times (X \text{ value}) + 8 \text{ cycles}
\]

where a value of A is 4 or greater and a value of X is 1 or greater. In PRGRNGE and ERARNGE, the CPUSPD value (which is frequency parameter) is loaded into A.

Because this routine is called from a jump table, three additional cycles are included in the above equation. Interrupts are not masked and COP is not serviced in DELNUS.
### Table 2. On-Chip FLASH Routines

<table>
<thead>
<tr>
<th>Routine Description</th>
<th>GETBYTE</th>
<th>RDVRRNG</th>
<th>PRGRNGE</th>
<th>Page Erase ERARNGE¹</th>
<th>Mass Erase ERARNGE</th>
<th>DELNUS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware Requirement</strong></td>
<td>Pullup on COMMPORT pin</td>
<td>Pullup on COMMPORT pin</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Entry Conditions</strong></td>
<td>COMMPORT direction is configured as input</td>
<td>H:X contains first address of range</td>
<td>LADDR contains last address read</td>
<td>A contains option selection, A=$00 for send-out option; A≠$00 for verify option</td>
<td>For send-out option, COMMPORT input and 0 data bit (DDRx0=0, PTx0=0)</td>
<td>For verify option, DATA contains data against which to compare read data</td>
</tr>
<tr>
<td><strong>Exit Conditions</strong></td>
<td>A is loaded with byte received</td>
<td>C bit is set if good compare; A contains checksum</td>
<td>DATA contains read FLASH data (verify option)</td>
<td>Preserves contents of H:X (address passed)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>I Bit</strong></td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>I bit is set</td>
<td>I bit is set</td>
</tr>
<tr>
<td><strong>COP</strong></td>
<td>Not Serviced</td>
<td>Not Serviced</td>
<td>Serviced except HLC908QT/QY</td>
<td>EY16 and GT16: Not serviced QT/QYxx: Serviced</td>
<td>EY16 and GT16: Not serviced QT/QYxx: Serviced</td>
<td>Not Serviced</td>
</tr>
<tr>
<td><strong>Subroutines Called</strong></td>
<td>—</td>
<td>—</td>
<td>DELNUS</td>
<td>DELNUS</td>
<td>DELNUS</td>
<td>—</td>
</tr>
<tr>
<td><strong>Variables Read</strong></td>
<td>LADDR, DATA</td>
<td>LADDR, DATA, CTRLBYT, CPUSPD</td>
<td>CTRLBYT, CPUSPD</td>
<td>CTRLBYT, CPUSPD</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Variables Modified</strong></td>
<td>DATA</td>
<td>DATA</td>
<td>DATA</td>
<td>DATA</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Stack Used</strong></td>
<td>4 bytes</td>
<td>6 bytes</td>
<td>7 bytes</td>
<td>5 bytes</td>
<td>5 bytes</td>
<td>3 bytes</td>
</tr>
</tbody>
</table>

**NOTES:**

¹ Before using this routine, confirm that this routine supports the user target device (Refer to Section 2, “Page Erase Issue”).
Device-Specific Information Related to On-chip FLASH Routines

4 Device-Specific Information Related to On-chip FLASH Routines

Table 3 shows the useful information for each device when the on-chip FLASH routines are used.

<table>
<thead>
<tr>
<th>Device</th>
<th>RAM</th>
<th>ROWSIZ</th>
<th>COMMPORT</th>
<th>FLBPR</th>
<th>Put_Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC68HC908EY16</td>
<td>$40</td>
<td>32</td>
<td>PTA0</td>
<td>$FF7E</td>
<td>—</td>
</tr>
<tr>
<td>MC68HC908GR4/8(A)</td>
<td>$40</td>
<td>32</td>
<td>PTA0</td>
<td>$FF7E</td>
<td>$FEAA</td>
</tr>
<tr>
<td>MC68HC908GT16</td>
<td>$40</td>
<td>32</td>
<td>PTA0</td>
<td>$FF7E</td>
<td>—</td>
</tr>
<tr>
<td>MC68HC908J88</td>
<td>$40</td>
<td>64</td>
<td>PTA0</td>
<td>$FE09¹</td>
<td>$FED6</td>
</tr>
<tr>
<td>MC68H(R/L)C908JK1(E)</td>
<td>$80</td>
<td>32</td>
<td>PTB0</td>
<td>$FE09(¹)</td>
<td>$FED0</td>
</tr>
<tr>
<td>MC68H(R/L)C908JK3E</td>
<td>$80</td>
<td>32</td>
<td>PTB0</td>
<td>$FE09(¹)</td>
<td>$FED0</td>
</tr>
<tr>
<td>MC68H(R/L)C908JL3(E)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC68HC908KX2/8</td>
<td>$40</td>
<td>32</td>
<td>PTA0</td>
<td>$FF7E</td>
<td>$FEAA</td>
</tr>
<tr>
<td>MC68HC908QT1/2/4</td>
<td>$80</td>
<td>32</td>
<td>PTA0</td>
<td>$FFBE</td>
<td>$FEA1</td>
</tr>
<tr>
<td>MC68HC908KT1/2/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC68HLC908QT1/2/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC68HLC908QY1/2/4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTES:
¹ FLBPR does not consist of FLASH. Unprotect the FLASH before programming or erasing.

In Table 3, RAM indicates the RAM start address. ROWSIZ indicates the size of a FLASH row. COMMPORT indicates a communication port for the monitor mode. FLBPR indicates FLASH block protect register address. Put_Byte indicates the address of the routine to send a byte through the communication port (COMMPORT). The communication baud rate is the same as the baud rate described in Section 3.1, “GETBYTE.” However, this routine is not officially supported. Therefore, the address might be changed in the future without any notice. We recommend that users develop their own Put_Byte routines.

5 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 Section 6, “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 bus frequency \( (f_{op}) \) 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 (bus) frequency, such that if \( f_{op} \) is 2.4576 MHz, then the value passed should be decimal 10 ($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 the control byte (CTRLBYT). Value $40 is set in this byte when calling ERARNGE to perform a mass erase. If ERARNGE is called with the intention of performing a page erase, then value $00 must be set.

### Table 4. Variables Used in Routines

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Size</th>
<th>Location/Passing Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>FADDR</td>
<td>First address of range of locations</td>
<td>2 bytes</td>
<td>H:X</td>
</tr>
<tr>
<td>LADDR</td>
<td>Last address of range of locations</td>
<td>2 bytes</td>
<td>Data structure</td>
</tr>
<tr>
<td>CPUSPD</td>
<td>( 4 \times f_{op} )</td>
<td>1 byte</td>
<td>Data structure</td>
</tr>
<tr>
<td>CTRLBYT</td>
<td>Mass bit (bit 6)</td>
<td>1 byte</td>
<td>Data structure</td>
</tr>
<tr>
<td>DATA</td>
<td>Data array</td>
<td>Variable</td>
<td>Data structure</td>
</tr>
</tbody>
</table>

### 6 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 RDVRRNG. 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

<table>
<thead>
<tr>
<th>Location</th>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM + $08</td>
<td>CTRLBYT</td>
<td>1 byte</td>
<td>Control byte setting erase size</td>
</tr>
<tr>
<td>RAM + $09</td>
<td>CPUSPD</td>
<td>1 byte</td>
<td>CPU speed passed as ( 4 \times f_{op} )</td>
</tr>
<tr>
<td>RAM + $0A</td>
<td>LADDR</td>
<td>2 bytes</td>
<td>Last address for read a range and program a range</td>
</tr>
<tr>
<td>RAM + $0B</td>
<td>DATA</td>
<td>Variable</td>
<td>Variable number of bytes of passed data for programming or verifying a block</td>
</tr>
</tbody>
</table>
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.

7 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>
<thead>
<tr>
<th>Addresses of Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6. Addresses of Routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETBYTE</td>
</tr>
<tr>
<td>MC68HC908EY16</td>
</tr>
<tr>
<td>MC68HC908GR4(A)</td>
</tr>
<tr>
<td>MC68HC908GT16</td>
</tr>
<tr>
<td>MC68HC908JB8</td>
</tr>
<tr>
<td>MC68HC908JL1(E)</td>
</tr>
<tr>
<td>MC68HC908JX2</td>
</tr>
<tr>
<td>MC68HC908Q1/2/4</td>
</tr>
<tr>
<td>MC68HC908QT1/2/4</td>
</tr>
<tr>
<td>NOTES:</td>
</tr>
<tr>
<td>1 This routine is used only for the mass erase operation.</td>
</tr>
</tbody>
</table>

8 MC68HC908KX8 Trim Routine

The MC68HC908KX8 Trim routine was supported in the original note. However, because another application note, AN2312, describes a more accurate trim method, this section was removed in this document.

9 Typical Routine Calls

This section provides examples of how the on-chip FLASH routines may be called.
9.1 Example for GETBYTE

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 and has a pullup.

GETBYTE:    equ   $2800       ;QY4 GETBYTE jump address
bclr  0,DDRA0     ;Configure port A bit 0 as an input
jsr   GETBYTE     ;Call GETBYTE routine
bcc   FrameError  ;If C bit is clear, framing error
; occurred. Take a proper action

9.2 Examples for RDVRRNG

This example calls the RDVRRNG routine to use the send-out option. In this example, a range of FLASH from $F000 to $F010 is read and read data is sent out through COMMPORT.

RDVRRNG:    equ   $2803       ;QY4 RDVRRNG jump address
bclr  0,PTA       ;Initialize data bit to zero PTA0=0
ldhx  #$F010      ;Load last address of range to
sthx  LADDR       ; LADDR
ldhx  #$F000      ;Load beginning address of range
; to H:X
clra              ;A=0 to select send-out option
jsr   RDVRRNG     ;Call RDVRRNG routine
; A contains a checksum value

The next example also uses the RDVRRNG. In this example, the verify option is selected. A range of FLASH from $E800 to $E81F is read and the read data is verified against data in the DATA array in RAM. When the verify is successful, the C bit in the CCR gets set.

RDVRRNG:    equ   $2803       ;QY4 RDVRRNG jump address
ldhx  #$0000      ;Index offset into DATA array
lda   #$AA        ;Initial data value to store in array
Data_load:   coma
sta   DATA,x      ;Fill DATA array, 32 bytes data,
; to verify against programmed FLASH
aix   #1          ; data (In this example verifying data
cphx  #$20        ; is $55, $AA, $55, $AA....)
bne   Data_load
ldhx  #$E81F      ;Load last address of range to
sthx  LADDR       ; LADDR
ldhx  #$E800      ;Load beginning address of range
; to H:X
lda   #$55        ;Write non-zero value to A to select
Typical Routine Calls

```assembly
; the verify option
jsr   RDVRRNG     ; Call RDVRRNG routine
bcc   Error       ; If bit C is cleared, verify failed
; Take a proper action
; A contains a checksum value
```

9.3 Example for Page Erase Operation

Before taking a look at this example, refer to Section 3.4, “ERARNGE — Page Erase,” to confirm that this operation is supported for the user target device.

This example performs the page erase operation using ERARNGE. The variable CPUSPD is set to a value which reflects an 8 MHz operating frequency, that is $4f_{op} = 4 \times 8 = 32$ (20). CTRLBYT must be loaded with $00$ to select the page erase operation. To erase a page from $EE00$ to $EE3F$, an address within this page must be loaded into H:X. Note that the FLASH block protect register must not be protecting the page.

```assembly
ERARNGE:    equ   $2806        ; QY4 ERARNGE jump address
mov   #$20,CPUSPD  ; fop = 8MHz in this example
mov   #$00,CTRLBYT ; Select Page erase operation
ldhx  #$EE01       ; Load any address within the
; page to H:X
jsr   ERARNGE      ; Call ERARNGE routine
```

9.4 Examples for Mass Erase Operation

This example performs the mass erase operation using the ERARNGE routine. CPUSPD is set to a value which reflects a 6 MHz operating frequency, that is $4f_{op} = 4 \times 6 = 24$ (18). CTRLBYT must be loaded with $40$ to select the mass erase operation. Any valid FLASH address is loaded into H:X when doing a mass erase. Note that the mass erase operation will not be successful if the FLASH block protect register has any block protected.

Regarding the HLC908QT/QYxx devices, note that only $f_{op} = 1$ MHz is supported.

```assembly
ERARNGE:    equ   $2806        ; QY4 ERARNGE jump address
mov   #$18,CPUSPD  ; fop = 6 MHz in this example
mov   #$40,CTRLBYT ; Select Mass erase operation
ldhx  #$F000       ; Load any address within the
; page to H:X
jsr   ERARNGE      ; Call ERARNGE routine
```

Regarding the JB8 and JL/JKxx(E) devices, FLBPR does not consist of FLASH. Because FLASH is always protected after reset, unprotect the entire FLASH array in user software before calling this routine.

Using MC68HC908 On-Chip FLASH Programming Routines, Rev. 3

12 Freescale Semiconductor
9.5 Examples for PRGRNGE

The first example shows how to program one full row. The variable CPUSPD is set to a value which reflects an 2.4576 MHz operating frequency, that is $4f_{op} = 4 \times 2.4576 = 10$ ($0A$). In this example, the target device is the MC68HC908QY4.

Regarding the HLC908QT/QYxx devices, note that only $f_{op} = 1$ MHz is supported.

```
PRGRNGE:    equ   $2809       ;QY4 PRGRNGE jump address
ldhx  #$0000      ;Index offset into DATA array
lda   #$AA        ;Initial data value (inverted)
Data_load:
coma              ;Alternate between $55 and $AA
sta   DATA,x      ;Fill DATA array, 32 bytes data,
; values to program into FLASH
aix   #1          ; (ie. 55, AA, 55, AA....)
cphx  #$20
bne   Data_load
mov   #$0A,CPUSPD ;fop = 2.4576 MHz in this example
ldhx  #$EE1F      ;Load last address of the row
sthx  LADDR       ; to LADDR
ldhx  #$EE00      ;Load beginning address of the row to H:X
jsr   PRGRNGE     ;Call PRGRNGE routine
```

The second example shows how to program one full page. The variable CPUSPD is set to a value which reflects an 3 MHz operating frequency, that is $4f_{op} = 4 \times 3 = 12$ ($0C$). However regarding the MC68HLC908QT/QY, PRGRNGE can program a range of FLASH locations that is up to 32 bytes and in the same row. Therefore, these devices can not use this example.

```
PRGRNGE:    equ   $2809       ;QY4 PRGRNGE jump address
ldhx  #$0000      ;Index offset into DATA array
lda   #$AA        ;Initial data value (inverted)
Data_load:
coma              ;Alternate between $55 and $AA
sta   DATA,x      ;Fill DATA array, 64 bytes data,
; values to program into FLASH
aix   #1          ; (ie. 55, AA, 55, AA....)
cphx  #$20
bne   Data_load
mov   #$0A,CPUSPD ;fop = 2.4576 MHz in this example
ldhx  #$EE3F      ;Load last address of the row
sthx  LADDR       ; to LADDR
ldhx  #$EE00      ;Load beginning address of the row to H:X
jsr   PRGRNGE     ;Call PRGRNGE routine
```

The second example shows how to program one full page. The variable CPUSPD is set to a value which reflects an 3 MHz operating frequency, that is $4f_{op} = 4 \times 3 = 12$ ($0C$). However regarding the MC68HLC908QT/QY, PRGRNGE can program a range of FLASH locations that is up to 32 bytes and in the same row. Therefore, these devices can not use this example.
Regarding the JB8 and JL/JKxx(E) devices, the FLBPR does not consist of FLASH. Therefore, an additional step is required to unlock the FLASH before calling the routine.

### 9.6 Example for DELNUS

This example shows how to use DELNUS. In this example a 100 µs delay is generated at $f_{op} = 4$ MHz using the DELNUS delay routine. To use this routine accurately, we need to calculate the value of X and keep track of the delays due to the setup instructions.

First, we need to know how many bus cycles a delay of 100 µs will take. This is simply delay time multiplied by bus frequency.

$$\text{Bus cycles} = 100 \, \mu s \times 4 \, \text{MHz} = 400 \, \text{cycles}.$$  

Next, we determine the value for the accumulator using the relationship:

$$A = CPUSPD = 4 \times f_{op} = 4 \times 4 = 16 \, (\$10)$$

Then we use the relationship:

$$\text{DELNUS} = 3 \times (A \, \text{value}) \times (X \, \text{value}) + 8$$

And the fact that we will use 9 bus cycles to setup the routine (lda, ldx, jsr):

$$\text{DEL}_{100} \, \mu s = 400 \, \text{cycles} = 9 + \text{DELNUS} = 9 + (3 \times 16 \times X + 8)$$

Solving for X:

$$X = \frac{(400 - 17)}{(3 \times 16)} = 8$$

$$A = 16 \, \text{and} \, X = 8 \, \text{are initialized before calling DELNUS.}$$

```assembly
DELNUS:equ $280C ;QY4 DELNUS jump address
DEL_100US:lda #$10 ;[2] A=16
                ldx #$08 ;[2] X=8
                jsr DELNUS ;[5] Call DELNUS routine
```

In this example, the total delay time is 9 setup cycles + (3 × 16 × 8 + 8) cycles = 401 cycles (100.25 µs).

### 10 Page Erase Workaround

This section provides a workaround for the page erase operation which described in Section 2, “Page Erase Issue.”

To use this workaround, the code in Section 12, “Workaround Code,” is programmed in FLASH. It is highly recommended that this routine is placed in an area of FLASH protected by the FLASH block protect register (FLBPR). Before calling this routine, CTRLBYT, CPUSPD, and H:X registers required for the page erase operation using the ERARNGE must be initialized properly. The code executes the following:

1. Copy a part of ERARNGE (page erase step 1 through step 6) located in ROM to RAM. The code is copied to address RAM+$0A and a total 72 bytes are copied from there.
2. Change a loop value to support $t_{\text{Erase}}=4\text{ms}$ in the code copied to RAM. This change guarantees minimum 10k program/erase FLASH endurance. However for the JB8, this change is not required.

3. Replace instruction "STA $FFFF" with instruction "LDA $FFFF" in the code copied to RAM. This change avoids writing to $FFFF$ due to the COP service.

4. Just after the copied routine, add a jump instruction which jumps to page erase step 7 in the original ERARNGE located in ROM.

5. Call the routine copied in RAM.

The following example shows how to execute the workaround routine.

```
mov   #$10,CPUSPD  ;CLEAR BIT 0 DATA DIRECTION
mov   #$00,CTRLBYT ;Select page operation
ldhx  #$F000
jsr   PageErase    ;Call a workaround routine
```

Regarding the JB8 and JL/JKxx(E) devices, FLBPR does not consist of FLASH. Therefore, an additional step is required to unlock the FLASH before calling the routine.

**NOTE**

COP is not supported in this workaround.

11 ROM Routines Source Code

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

PURPOSE:
GET A BYTE OF DATA ON PTA0. 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.

NOTES:
CYCLE PATH FOR EACH BIT RECEPTION MUST BE KEPT THE SAME TO MAINTAIN A STEADY BAUD RATE. IF RESULT IS GOOD, THEN Acc = BYTE RECEIVED. PORT A0 CONFIGURED AS AN INPUT.

Figure 1. GETBYTE
RDVRRNG

STORE DESTINATION IN TEMP1

INIT TEMP2 = FF AS COMPARISON STATUS

INIT TEMP0 = FF AS INDEX INTO DATA

GET FLASH DATA FROM FADDR

DESTINATION = SERIAL?

VERIFY FLASH DATA = INPUT DATA

STORE FAILURE ($7E) INTO TEMP2

WRITE FLASH DATA INTO DATA

ACUMULATE CHECKSUM

INC FADDR

INC TEMP3 FOR DATA POINTER

RDVRRNG010

YES

FADDR = LADDR+1?

YES

GET PASS/FAIL FROM TEMP2

GET CHECKSUM

RETURN

NO

RDVRRNG020

CALL PUT_BYTE

RDVRRNG030

RDVRRNG

PURPOSE:
READ AND/OR VERIFY A RANGE OF FLASH MEMORY

NOTES:
H:X CONTAINS THE FIRST ADDRESS OF THE RANGE;
LADDR CONTAINS THE LAST ADDRESS TO BE READ;
Acc CONTAINS THE DESTINATION OF THE FIRST BYTE OF THE READ DATA (0 = PTA0);
DATA CONTAINS THE DATA TO COMPARE THE READ DATA AGAINST

Figure 2. RDVRRNG

Using MC68HC908 On-Chip FLASH Programming Routines, Rev. 3
**PURPOSE:**

Programs a range of addresses in flash memory. Allows programming of a range of addresses, which does not have to be on page boundaries, either beginning or end. For example, programming $F001 to $F008 is valid. This is to prevent trying to program a non-flash address and getting back a bad verification.

**NOTES:**

- H:X contains the first address in the range;
- CTRLBYT specifies the programming mode;
- LADDR contains the last address to be read;
- DATA contains the data to be programmed;
- CPUSPD contains the CPU speed for delay accuracy.
**DELNUS**

**PURPOSE:**
DELAY FOR \( N \times 12 \mu s \) FOR \( f_{op} \geq 1 \text{ MHz} \);

**NOTES:**
- \( D = (\text{DELAY TIME } [\mu s] \div 12) \) IN X, \( C = (f_{op} [\text{MHz}] \times 4) \)
- \( \text{IN Acc CYCLES} = 5 + (\text{DELAY } + 12) \times 3(4f_{op} - 3) + 9 = 5 + \text{DELAY } \times f_{op} \)
- X CONTAINS THE TIME = 12 OF DELAY (IN \( \mu s \));
- Acc CONTAINS CPUSPD (CPU SPEED \( \times 4 \));
- CPU SPEED MUST BE \( \geq 1 \text{ MHz} \)

---

Figure 4. DELNUS
ERARNGE

STACK ADDRESS PASSED IN H:X

SET ERASE BIT IN FLCR

MASSBIT SET IN CTRLBYT?

YES

SET MASS BIT IN FLCR

READ THE BLOCK PROTECT REGISTER

MASSBIT SET IN CTRLBYT?

NO

WRITE CONTENT OF Acc TO ADDRESS SPECIFIED IN H:X

CALL DELNUS WITH TNVSQ IN H:X AND CPUSPD IN Acc

SET ERASE AND MASS BITS IN FLCR

RETURN

YES

SET DELAY LOOP CNTR FOR 5

CALL DELNUS WITH TERRSEQ IN H:X AND CPUSPD IN Acc

BUMP COP

LOOP COUNTER = 0?

YES

BUMP COP

NO

MASSBIT SET IN CTRLBYT?

SET DELAY LOOP CNTR FOR 20

CALL DELNUS WITH TNVHQ IN H:X AND CPUSPD IN Acc

CLEAR HVEN BIT IN FLCR

RESTORE ADDRESS PASSED FROM STACK TO H:X

RETURN

NO

YES

WRITE TO THE BLOCK PROTECT REGISTER

CALL DELNUS WITH TNVHLQ IN H:X AND CPUSPD IN Acc

 Massbit SET IN CTRLBYT?

NO

DECREMENT LOOP COUNTER

NO

YES

Massbit SET IN CTRLBYT?

CLEAR HVEN BIT IN FLCR

Massbit SET IN CTRLBYT?

YES

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?

NO

Massbit SET IN CTRLBYT?
* 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

* EQUATES

* PROGRAMMING TIMES IN us
* 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

Using MC68HC908 On-Chip FLASH Programming Routines, Rev. 3
* 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.
* BITTIMING = 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 ;1 bit into Acc
RORA ;3 get next bit

BCC GBIT ;baud calculation

STOPBIT: ;look for stop bit

JSR GET_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 f_COP = 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
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 ;(E')
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

************************************************************************************
Using MC68HC908 On-Chip FLASH Programming Routines, Rev. 3

ROM Routines Source Code

CPHX LADDR ;COMPARE SOURCE ADDR TO THE LAST ADDRESS
BHS NOMO ;IF NOT YET DONE, LOOP FOR ANOTHER
AIX #1 ;INCREMENT SOURCE ADDRESS
BRA RDVRNRNG010
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
Using MC68HC908 On-Chip FLASH Programming Routines, Rev. 3

ROM Routines Source Code

PRGFP1:

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-ERASE.-ERASE.)
STA  FLCR ;WRITE THIS TO THE FLASH CONTROL REG.

PRGFP2 LDA  FLBPR ;READ FROM BLOCK PROT. REG.

PRGFP3:

  IFEQ  TESTMOD
  LDA  ,X
  ENDF
  IFNE  TESTMOD
  STA  ,X ;WRITE TO ANY FLASH ADDRESS WITHIN THE ROW
  ENDF
  PSHH ;(D)
  PSHX ;(E)

PRGFP4 LDX  #TNVSQ ;DELAY FOR TNVS
LDA  CPUSPD
BSR  DELNUS

PRGFP5 LDHX  #FLCR ;SET THE HVEN BIT IN FLCR
LDA  ,X
ORA  #HVEN. 
STA  ,X

PRGFP6 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
************************************************************************************

PRGFP7 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
Using MC68HC908 On-Chip FLASH Programming Routines, Rev. 3

ROM Routines Source Code

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.
  CPHX LADDR ;TO LAST ADDR
  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 $f_{OP} \geq 1$ MHz, Delay $\geq 12$ ms
* (delay time[µs]/12) in H:X, ($f_{OP}$[MHz]*4) in Acc
* If $f_{OP} > 1$ then
* CYCLES = 5+Delay/12[3(4f_{OP}-3)+9] = 5+DELAY*$f_{OP}$
* If $f_{OP} = 1$ then CYCLES = 5+12(Delay/12) = 5+DELAY
* where delay in µs and $f_{OP}$ in MHz
************************************************************************************

DELNUS: DECA ;1 CYCLE
NXTX PSHA ;2
DECA ;1
DECA ;1
DBNZ A *
PULA ;2
**ERARNGE**

SEI
PSSHH  ;KEEP ADDRESS PASSED
PSHX

CLRA ;SET ERASE BIT, AND
ORA #ERASE.
BRCLR MASSBIT,CTRLBIT,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
    IPNE 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
Workaround Code

;*********************************************************************
;* NAME: PageErase
;* Assembler: P&E Microcomputer Systems Casm08
;* PURPOSE:
;* This routine is used for erasing a page size of FLASH.
;* In this routine, a part of ERARNGE routine located in ROM is
;* copied into RAM and then execute the routine out of RAM.
;* To work a page erase operation properly, set proper values at
;* Setup 1 - 4.
;* ENTRY CONDITIONS:
;* H:X contains an FLASH address within a page to be erased
;* Bit 6 in CTRLBYT is cleared
;* CPUSPD contains an integer of 4 x bus frequency [MHz]
;* COP must be disabled
;* EXIT CONDITIONS:

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 ;PULL INCREMENT CNTR OFF STACK
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 #TNVH ;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 ;THOSE 3 INST. DELAY FOR
RTS ;AT LEAST 1 μs (TRCV)

*********************************************************************
Workaround Code

;* I bit is set
;* The contents of H:X (address passed) is preserved
;* SUBROUTINES CALLED: DELNUS
;* VARIABLES READ: CTRLBYT, CPUSPD
;* STACK USED: 10 bytes (including the call to this routine)
;* SIZE OF THIS CODE:
;* 908GR4/8, 908KX2/8, 908JK1(E), 908JL/JK3(E) - 33 bytes
;* 908JB8 - 29 bytes
;* Note:
;* a. Locations from RamStart+$0A to RamStart+$54 is used in this
;*    routine, where RamStart is a start address of RAM.
;* b. Since the COP is not supported, it must be disabled.
;*********************************************************************
; Setup 1: Device Selection
; Distinguish between HC908JB8 and other parts
;*********************************************************************
$SETNOT     JB8               ;This example does not select JB8
;* $SET        JB8            ;Use this setup if a part is JB8
;*********************************************************************
; Setup 2: FLASH start address (locFLASH)
; Specify a start address where this routine is placed in FLASH.
; We highly recommend that this routine is protected by FLBPR.
;*********************************************************************
locFLASH:   equ   $FDC0
;*********************************************************************
; Setup 3: RAM start address (RamStart)
; For 908GR4/8, 908KX2/8, 908JB8
;* RamStart:   equ   $0040
; For 908JK1(E), 908JL/JK3(E)
;* RamStart    equ   $0080
;*********************************************************************
RamStart:   equ   $0040       ;This example for the HC908GR8
;*********************************************************************
; Setup 4: DELNUS start address (DELNUS)
; For 908GR4/8
;* DELNUS:     equ    $1D96
; For 908KX2/8
;* DELNUS:     equ    $12C3
; For 908JK1(E), 908JL/JK3(E), 908JB8
;* DELNUS:     equ    $FD21
;*********************************************************************
DELNUS:     equ   $1D96       ;This example for the HC908GR8
;*********************************************************************
DATSTRC:    equ   RamStart+8  ;data array starts from RamStart
CodeSize:   equ   $48         ;72 byte code to be copied to RAM

;* Variable locations used in this routine.
   org   DATSTRC

Using MC68HC908 On-Chip FLASH Programming Routines, Rev. 3
CTRLBYT: rmb 1 ;control byte for erase size
CPUSPD: rmb 1 ;CPU speed
codeRAM: rmb CodeSize ;a part of the ERARNGE is copied to
                  ;this location.
CodeJmp: rmb 3 ;jump to ROM routine

ramERARNGE: equ codeRAM+$A ;ERARNGE start address in RAM
locECalls: equ codeRAM+$37 ;location where ECalls is initialized
locCOP: equ codeRAM+$39 ;location where COP is serviced
jmpAddr: equ DELNUS+$4B ;jump location to a part of the ERARNGE
ECalls: equ $14 ;support 4ms Terase
exLDA: equ $C6 ;extended LDA instruction
exJMP: equ $CC ;extended JMP instruction

org locFLASH

PageErase:
pshx ;save H and X values to stack
pshh
ldhx #CodeSize ;initialize pointer
Code2RAM:
lda DELNUS-1,x ;copy DELNUS and a part of ERARNGE
sta codeRAM-1,x ; into RAM
dbnzx Code2RAM ;decrement pointer and loop back
                 ; until done
$IFNOT JB8
                 ;if JB8 is selected in Setup 1, two
                 ; instructions below are not included
lda #ECalls     ;change 1ms to 4ms for Terase
sta locECalls
$ENDIF

lda #exLDA ;change "sta $FFFF" to "lda $FFFF"
sta locCOP ; in ERARNGE
mov #exJMP,CodeJmp ;add "jmp ROM_step7" just after copied
ldhx #jmpAddr ; code
sthx CodeJmp+1
pulh ;set a location within a page to be
pulx ; erased at H:X
jsr ramERARNGE ;execute a page erase operation from RAM
rts