Using FLASH as EEPROM on the MC68HC908GP32

By Derrick H.J. Klotz
Field Applications Engineer
Toronto, Canada

Introduction

This application note describes a method for using the on-chip FLASH memory of the MC68HC908GP32 as one would typically use EEPROM (electrically erasable programmable read-only memory). For the purposes of this application note, the FLASH memory that is manipulated via this method is referred to as "FlashEE." It is expected that the reader is somewhat familiar with the MC68HC908GP32/08GP32 Technical Data book, Motorola document order number MC68HC908GP32/H, as well as typical EEPROM device usage.

In many projects, EEPROMs are used as nonvolatile storage for calibration data, control information, data logging, etc. The FLASH memory technology employed in the M68HC908 Family of microcontrollers (MCU) is capable of being reprogrammed easily while the application software is executing. Project cost savings can be realized by simply using an appropriate section of the on-chip FLASH memory as one would use an off-chip EEPROM device.
In order to verify the correct operation of the FlashEE routines, a test program is included in this application note. This program demonstrates how these routines would be integrated into a project. The test program communicates with a host computer via a simple RS-232 connection in order to facilitate testing. User programs that include the FlashEE routines are free to implement them as one would commonly expect to use EEPROM algorithms for various forms of data storage.

**FlashEE Implementation**

Specifications for the M68HC908 Family FLASH memory indicate a program/erase cycle endurance of 10,000 cycles across the full operating temperature range. This is typically more than sufficient for most applications. However, this value can be mathematically elongated by the number of times a small data block can fit within a FLASH erase page.

It is important to note that the while the FLASH memory is “page” erasable, it is, in fact, byte programmable. Its organization is configured for efficient “row” programming, but any algorithm must write each location on a byte-by-byte basis. There is no limit to the minimum number of bytes that must be programmed within a row. But every time a row is programmed, the high voltage charge pump must be enabled and disabled, regardless of the number of bytes that are programmed.

A critical FLASH memory specification is the cumulative program high voltage period. This is the maximum total time that a specific FLASH program row is subjected to being programmed between erasures. The technical data for the MC68HC908GP32 lists this maximum specification as being 4 ms.

The subroutines herein are optimized to program relatively small blocks of data sequentially within the FLASH memory, thereby minimizing the total cumulative length of time spent programming. These subroutines will only erase the FlashEE space when there is no more room for the next data block. Each FLASH erase page consists of two program rows. The FLASH erase page size for the MC68HC908GP32 is 128 bytes.
If an example application uses a FlashEE data block size of four bytes, then 32 program cycles of four bytes each will be performed prior to executing a single erase cycle, at which point this would be considered one complete program/erase cycle. Hence, in this example, the FlashEE endurance would be calculated as being 320,000 program/erase cycles (in other words, 10,000 x 32).

The high voltage program time used by the subroutines herein to program four bytes is less than 150 $\mu$s. Each program row would be subjected to a program cycle 16 times (remember, two program rows per erase page). This provides a cumulative time of less than 2.4 ms (150 $\mu$s x 16), which is less than the maximum specification given.

The FLASH block protect register (FLBPR) points to the first FLASH memory location to be protected. When programmed, every location from that address to $FFFF$ will be protected from accidental erasure. For this reason, it is easiest to assign the FlashEE sections to the very beginning of the FLASH memory, assuming that the FLBPR may be used to protect application code.

To simplify the FlashEE implementation, some essential guidelines have been used, specifically:

- FlashEE data is written in blocks of multiple bytes.
- Each FlashEE data block fits within a single FLASH program row.
- The first FlashEE data block byte to be written cannot be equal to $FF$.

Programming only one byte is possible, but may result in application code inefficiencies since an entire 128-byte FLASH erase page would need to be reserved for each single byte of FlashEE.

If the data storage space requirement exceeds the size of a single FLASH program row (i.e., more than 64 bytes on the MC68HC908GP32), then the data will need to be split up over multiple FlashEE sections, each occupying one FLASH erase page (i.e., 128 bytes on the MC68HC908GP32). If multiple FlashEE sections are used, they must be manipulated separately. This is demonstrated with the test program provided.
As part of a search algorithm, the software interrogates the first byte location of each FlashEE data block to see if it is erased (i.e., $FF). For this reason, when writing a new block of data, the first byte must not be $FF. All subsequent data block locations have no restrictions on their stored value.

**Figure 1** shows the MC68HC908GP32 memory map and the space occupied by two separate FlashEE sections (128 bytes each). Note that the erased state of the FLASH memory is $FF and that the first 10 locations ($8000 to $8009) are not erased. This represents the current state of FlashEE1 at the end of the example host test session shown in **Figure 2** and described later in this text.

As defined in the example provided, FlashEE1 starts at address $8000 with a data block size of five bytes. FlashEE2 starts at address $8080 and has a data block size of seven bytes. Although reasonably obvious, delineations are shown in **Figure 1** that demonstrate that neither five nor seven evenly fit into a FLASH program row that is 64 bytes in size. The routines provided will store the appropriate data sequentially in the respective FlashEE sections only up to the last complete available data block space. As defined, FlashEE1 will never store a value in the last four locations of its FLASH program row (i.e., locations $803C to $803F and $807C to $807F will always be erased.) Similarly, the last byte in each FLASH program row of FlashEE2 will also remain erased (i.e., locations $80BF and $80FF). A FlashEE section with a data block size of 33 to 64 bytes inclusive will only fit the data into each FLASH program row once, leaving the remaining locations erased.

The program files provided are:

- **flashee.equ** — FlashEE subroutine operational parameter definitions
- **flashee.asm** — FlashEE subroutines
- **eetest.asm** — FlashEE test program
- **gp32.equ** — MC68HC908GP32 microcontroller definitions
The image contains a memory map for the MC68HC908GP32 microcontroller, showing the layout of the I/O registers, RAM, Flash memory, and control registers. The memory map is divided into sections for FlashEE1, FlashEE2, I/O registers, RAM, and Flash memory. Each section is represented with hexadecimal addresses and values.

### I/O Registers
- **FlashEE1**: Addresses 0000 to 000F, 0020 to 002F
- **FlashEE2**: Addresses 0000 to 000F, 0020 to 002F

### RAM
- **FlashEE1**: Addresses 0040 to 004F
- **FlashEE2**: Addresses 0040 to 004F

### Flash Memory
- **FlashEE1**: Addresses 0080 to 00FF
- **FlashEE2**: Addresses 0080 to 00FF

### FE00 - Control Registers and Monitor ROM
- **FFDC**: Addresses 0000 to 000F

### Vectors
- **FFFF**: Addresses 0000 to 000F

**Figure 1.** MC68HC908GP32 Memory Map and FlashEE Implementation

For more information on this product, go to: www.freescale.com

AN2183
flashee.equ Subroutines

The FlashEE subroutines are tailored to the needs and operation conditions of a specific application by modifying the parameters provided in this file, which must be included near the top of the application program file. To help avoid assembly language label naming conflicts, all labels used in this file start with either “EE” or “Ram”.

The parameters defined are grouped into three specific categories:

- Microcontroller FLASH memory parameters
- FlashEE data parameters
- Microcontroller bus frequency parameters

Microcontroller FLASH Memory Parameters

The specific sizes of the FLASH memory program row and erase page are defined here (64 bytes and 128 bytes, respectively, for the MC68HC908GP32). Also, the erased state of a FLASH byte is also declared as $FF.

These parameters are initially defined as:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE_FlashPage</td>
<td>equ</td>
</tr>
<tr>
<td>EE_FlashRow</td>
<td>equ</td>
</tr>
<tr>
<td>EE_FlashErased</td>
<td>equ</td>
</tr>
</tbody>
</table>

FlashEE Data Parameters

The starting address of each FlashEE section is declared here. These parameters are not directly used by the FlashEE subroutines. They are passed forward by the application program when these subroutines are called. For demonstration purposes, two separate FlashEE sections are shown, although many more are possible.

The size of each FlashEE data block is also defined. Note that each FlashEE section can have its own unique data block size.
The parameters for FlashEE1 are initially defined as:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE_StartAddr1</td>
<td>$8000</td>
</tr>
<tr>
<td>EE_BlockSize1</td>
<td>5</td>
</tr>
</tbody>
</table>

The parameters for FlashEE2 are initially defined as:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE_StartAddr2</td>
<td>$8080</td>
</tr>
<tr>
<td>EE_BlockSize2</td>
<td>7</td>
</tr>
</tbody>
</table>

Note that the assembler will assign the label "EE_StartAddr2" with a value of $8080.

The FlashEE subroutines employ delay loops that must be tuned to the microcontroller bus frequency. The values for these parameters for a bus frequency of 7.3728 MHz are provided in the software listing. If a different bus frequency is desired, then these parameters must be changed in accordance with the formula provided, specifically:

\[
\text{Value} = \left( (\text{delay in } \mu\text{s}) \times (\text{bus frequency in MHz}) - 2 \right) \div 3
\]

As an example, a delay of 30 \(\mu\text{s}\) with a bus frequency of 7.3728 MHz would result in the following:

\[
\text{RamDelay30} = \left( (30) \times (7.3728) - 2 \right) \div 3 = 73.06 \approx 74
\]

For a 7.3728-MHz bus frequency, the required parameters are initially defined as:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RamDelay5</td>
<td>12</td>
</tr>
<tr>
<td>RamDelay10</td>
<td>24</td>
</tr>
<tr>
<td>RamDelay30</td>
<td>74</td>
</tr>
<tr>
<td>RamDelay50</td>
<td>122</td>
</tr>
</tbody>
</table>

**Microcontroller Bus Frequency Parameters**

**flashee.asm Subroutines**

This file contains the FlashEE subroutine source code and must be included in the application program file. These subroutines only use local variables on the stack and, hence, do not require any other predefined global variable space resources. The maximum available stack space...
requirement is 80 bytes. To help avoid assembly language label naming conflicts, all labels used start with either "EE" or "Ram". The flow diagrams of these routines (with the exception of the Dump utility, due to its simplicity) are provided in Figure 3 through Figure 8.

The available FlashEE routines are:

- **EERead** — Read the current valid FlashEE data block
- **EEErase** — Erase an entire FlashEE section
- **EEWrite** — Write a new FlashEE data block

### EERead

This subroutine is called with the 16-bit index register H:X pointing at the starting address of the desired FlashEE section and the accumulator preloaded with that FlashEE section’s data block size. The FlashEE section is sequentially scanned, block by block, until an erased FLASH byte (i.e., $FF) is found occupying the first location of a data block or the end of the section is reached.

This subroutine returns with the 16-bit index register H:X pointing to the first location of the most recent FlashEE data block and the data in that location stored in the accumulator. The calling routine should check the accumulator value for $FF to see if any data has been stored in the FlashEE. Refer to **Check_Read** subroutine in the `eetest.asm`. This check should only be needed when an EERead is performed before the very first EEWrite.

### EEErase

This subroutine is used to erase the contents of a FlashEE section. It is called with the 16-bit index register H:X pointing at the starting address of the desired FlashEE section. The value in H:X is returned unchanged. Regardless of bus frequency, this subroutine will execute in just over one millisecond. (Specifically, at a bus frequency of 7.3728 MHz, this subroutine executes in about 1096 µs.)

Note that this subroutine will copy and execute a program in RAM. This is required due to the fact that erasing FLASH locations cannot be performed by code being executed from the FLASH memory. While executing code from RAM, all interrupts are disabled.
EEWrite

This subroutine requires two address pointers when called. The 16-bit starting address of the desired FlashEE section must be saved on the stack just prior to calling this subroutine. In addition, the 16-bit index register H:X must point to the first byte of a source data block, which is typically a buffer located in RAM. As well, the accumulator is preloaded with the FlashEE data block size. This subroutine will then copy the required number of bytes sequentially from the source location into the FlashEE.

As a point of reference toward understanding the speed of this subroutine, consider using a data block of 16 bytes and a bus frequency of 7.3728 MHz. The best case programming time would be about 644 µs. The worst case would involve writing the data block into the FlashEE with no available room. This requires that the FlashEE first be erased prior to programming the data. In this given scenario, this subroutine executes in about 1772 µs, of which just over 1000 µs is due to the FlashEE erasure procedure.

Note that this subroutine will copy and execute a program in RAM. This is required due to the fact that programming FLASH locations cannot be performed by code being executed from the FLASH memory. While executing code from RAM, all interrupts are disabled.

eetest.asm Subroutines

This file provides an executable demo program used to test and verify the use of the FlashEE subroutines and files. Note that the files flashee.equ, flashee.asm, and gp32.equ are incorporated in this test program via the "include" assembler directive.

A simple user interface is provided via the on-chip serial communications interface (SCI) and industry standard RS232 communications with a host computer executing a simple terminal program. The serial bit rate is configured for 115.2 kbaud.

The software implements four single ASCII character commands (case insensitive) and provides the ability to test and interrogate two separate...
FlashEE sections. Each command is followed by either "1" or "2" to indicate the desired FlashEE section.

The commands are:

- **R** — Read FlashEE
- **E** — Erase FlashEE
- **W** — Write FlashEE
- **D** — Dump FlashEE

An example of the host terminal display during a simple test session using FlashEE1 with a block size of five bytes is shown in Figure 2. The program implements a default prompt to the user of FlashEE>.

The first user command is **d1**, or "dump the contents of FlashEE1." All 128 bytes are printed out in a format that is relatively easy to read. Note that at this point, the entire FlashEE section is erased (i.e., all locations contain $FF). The second command issued is **r1**, or "read the most recent FlashEE1 data block." Since FlashEE1 is erased, a message is sent indicating that no data is currently stored in FlashEE1.

Next, the user has requested that a block of data be written into FlashEE1 with **w1**. The program responds with an equal sign (=) and then accepts the correct number of hexadecimal values (five in total for FlashEE1). The user verifies that the data has been programmed correctly by first using the **r1** command, and secondly with **d1** which clearly indicates where the data has been stored.

The user enters another block of data with the next **w1** command. The following **d1** entry shows that the new data has been correctly stored in the next data block location. And the next **r1** command confirms that the read subroutine correctly identifies which is the most recent data.

Continuing with this method, the demo program provided was used to test and verify the correct operation of the FlashEE subroutines.
Figure 2. Simple Host Terminal Test Session Display

**Read FlashEE (R)**
After entering the single character R, followed by either "1" or "2", the program will respond by outputting the most recent data byte block. In the example shown, the first time R is entered and the FlashEE is erased as indicated by the software’s response. Subsequent examples show the current data block data being reported back to the host.

**Erase FlashEE (E)**
This command will directly execute the EEERase subroutine and erase the selected FlashEE section.
Write FlashEE (W) After entering the single character W, followed by either "1" or "2", the program will respond by prompting the user to enter a data block string of data. Each entry must be a valid hexadecimal value and is echoed back as typed. Entering a non-hexadecimal value results in the process being stopped with the default prompt being output. After entering the required number of bytes, the FlashEE is programmed with the data provided.

Dump FlashEE (D) This command will output the current contents of the selected FlashEE section.

Summary

Most projects using the M68HC908 Family of microcontrollers requiring nonvolatile data storage that is application software programmable need not incur the added cost of external EEPROM devices. A method for using the on-chip FLASH memory for such application purposes has been demonstrated with this application note.
Figure 3. eetest.asm Software Flow Diagram
Figure 4. Read FlashEE Software Flow Diagram
Figure 5. Erase FlashEE Software Flow Diagrams
Figure 6. Write FlashEE Software Flow Diagram
Figure 7. Write FlashEE RAM Routine Software Flow Diagram
Figure 8. Find First Erased FlashEE Data Block Software Flow Diagram
flashee.equ

;******************************************************************************
;* Flash as EEPROM - MC68HC908GP32                                         *
;******************************************************************************

;* This file provides the application specific parameters for the FlashEE routines.
;* This program has been specially tailored towards the MC68HC908GP32.

;******************************************************************************

;* File name: flashee.equ       Current Release Level: 1.0         *
;* Last Edit Date: 15-Jun-01     Classification: ES          *
;* Include Files: gp32.equ       : MC68HC908GP32 MCU definitions    *
;* Assembler: P&E’s CASM08Z     Version: 3.16                      *
;* Target: MC68HC908GP32        *
;* Documentation: MC68HC908GP32/H Rev 3                            *
;* Motorola Microcontroller Technical Data                           *

;* Author: DHJ Klotz
;* First Release: 15-Jun-01

;* Update History:

;* Rev Date Author Description of Change
;* ------ ------- ------ ----------------------------------
;* ES 1.0 15-Jun-01 DHJK Initial release.

;* Notes:
;* - This file is intended to be included within another source program along with
;* the program file "flashee.asm". All labels used in this file start with either
;* "EE" or "Ram".
;* - The "FlashEE Data Parameters" and "Microcontroller Bus Frequency Parameters"
;* declared at the start of this listing must be tailored to the specific needs of
;* the target application prior to using the programs herein.

;******************************************************************************

;* Motorola reserves the right to make changes without further notice to any product
;* herein to improve reliability, function, or design. Motorola does not assume any
;* liability arising out of the application or use of any product, circuit, or software
;* described herein; neither does it 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 life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such intended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

Motorola and the Motorola logo are registered trademarks of Motorola Ltd. and its subsidiaries.

Freescale Semiconductor, Inc.

For More Information On This Product, Go to: www.freescale.com
flashee.asm

;********************************************************************************************
;*                                                                                          *
;*  Flash as EEPROM - MC68HC908GP32                                                        *
;*                                                                                          *
;*  This file provides the low level assembly routines for using the Flash as EEPROM.       *
;*  This program has been specially tailored towards the MC68HC908GP32.                     *
;*                                                                                          *
;********************************************************************************************

;* File name:          flashee.asm                     Current Release Level:      1.0     *
;* Last Edit Date:     15-Jun-01                       Classification:             ES      *
;* Include Files:      gp32.equ          : MC68HC908GP32 MCU definitions *
;*                        flashee.equ       : FlashEE parameters *
;* Assembler:          P&E’ s CASM08Z          Version:    3.16 *
;* Target:             MC68HC908GP32             *
;* Documentation:      MC68HC908GP32/H Rev 3     *
;*                      Motorola Microcontroller Technical Data *

;* Author:             DHJ Klotz                  *
;* First Release:      15-Jun-01          *

;* Update History:
;* Rev     Date       Author  Description of Change
;* ------  ---------  ------  ----------------------------------------------------------- *
;* ES 1.0  15-Jun-01  DHJK     Initial release.                             *

;* Notes:                                                                                  *
;* - This file is intended to be included within another source program along with        *
;*   the FlashEE parameter file "flashee.equ". All labels used in this file start with    *
;*   either "EE" or "Ram".                                                               *
;* - The "FlashEE Data Parameters" and "Microcontroller Bus Frequency Parameters"         *
;*   declared in the include file "flashee.equ" must be tailored to the specific needs    *
;*   of the target application prior to using the programs herein.                        *
;* - Callable subroutines are:                                                          *
;*   EERead : returns with H:X pointing to first FlashEE data block entry                 *
;*   EEWrite : programs FlashEE with data block pointed to by H:X                         *
;*   EErase : erases entire FlashEE space pointed to by H:X                              *

AN2183
Read FlashEE Block Subroutine

This subroutine expects H:X to be pointing to the first FlashEE page location and returns with H:X pointing to the most recent FlashEE data. The FlashEE data block size is passed forward in ACC.

Calling convention:

ldhx #EE_StartAddr
lda #EE_BlockSize
jsr EERead

Returns: H:X -> FlashEE block address
ACC = first FlashEE data byte

Changes: everything

EERead:

Find first erased location within FlashEE page.

Check for room within first FlashEE row.
If the entire row is erased, then return with H:X unchanged. If the next available erased FlashEE block is within the first program row, then return pointing to the block just before it.

psaha ; save FlashEE data block size
pshx ; save FlashEE address lsb on stack
bsr EEFindFirst ; go find next free location
cpx 1,sp ; check if start of 1st row is 1st blank location
beq EERead3 ; exit if so
cmp 2,sp ; check if there’s room for another data block
bpl EERead2 ; exit if so

Check for room within second FlashEE row.
txa          ; save current address lsb
pulx         ; restore FlashEE address lsb
aix          ; H:X now points to second program row
ps ha         ; save previous address lsb on stack
lda ,x       ; check if first location
cmp #EE_FlashErased ; is erased
bne EERead1  ; skip if so
pulx         ; else, restore previous FlashEE address lsb
ps hx         ; put back on stack for clean exit
bra EERead2  ; and exit

EERead1:
lda 2,sp     ; get FlashEE data block size
bsr EEFindFirst ; go find next free location

EERead2:
sub 2,sp     ; 16-bit subtraction
sub         ; in order
tax         ; to adjust
pshh         ; FlashEE pointer
sbc #0       ; backwards
psha         ; one data block
pulh         ; size

EERead3:
ais #2       ; deallocate stack usage
lda ,x       ; get first FlashEE data byte
rts          ; return

;* RAM Executable FlashEE Page Erase Subroutine -----------------------------------------------
;* This subroutine will erase the Flash memory page that is being pointed to by H:X.
;* This subroutine is copied into and executed from RAM and expects to be called via
;* "jsr ,x".
;*
RamEraseEE:
;
; Retrieve page address.
;
ps ha          ; save previous CCR on stack
lda (RamEraseSize),x  ; get msb
ldx (RamEraseSize+1),x ; and lsb of address
psha          ; put msb
pulh          ; into HREG
;
; Step 1:
; Set ERASE, read the Flash Block Protect Register and write any data into Flash page.
;
lda #(ERASE)    ; set ERASE control bit
sta flcr       ; in Flash Control Register
lda flbpr      ; read from Flash Block Protect Register
sta ,x         ; write any data to address within page
; Step 2:
; Wait for >10us, then set HVEN.
;
 lda     #RamDelay10                 ; wait
dbnza   *                           ; for 10us
 lda     #{ERASE | HVEN}             ; set HVEN control bit
 sta     flcr                        ; in Flash Control Register
;
; Step 3:
; Wait for >1ms, then clear ERASE.
;
 ldx     #20                         ; outer loop
 RamEraseEE1:                                ; set for 20x
 lda     #RamDelay50                 ; inner loop
dbnza   *                           ; set for 50us
dbnzx   RamEraseEE1                 ; loop back until done
 lda     #{HVEN}                     ; clear ERASE control bits
 sta     flcr                        ; in Flash Control Register
;
; Step 4:
; Wait for >5us, then clear HVEN, then wait >1us and return.
;
 lda     #RamDelay5                  ; wait
dbnza   *                           ; for 5us
cira    ; clear HVEN control bit
 sta     flcr                        ; in Flash Control Register
 pula    ; retrieve previous CCR
 brn     *                           ; wait for at least 1us before
 rts     ; returning

RamEraseSize:   equ     {*-RamEraseEE}

;* FlashEE Page Erase Subroutine  =============================================================
;* This subroutine will erase the Flash memory page that is being pointed to by H:X.
;* 60 bytes of stack space is used, including this subroutine's call return address.
;* Calling convention:
;*  ldhx    #EE_StartAddr
;*  jsr     EEErase
;*  Returns: H:X unchanged
;*  Changes: ACC
;* EEErase:
;  pshx                     ; save pointer
  pshh                     ; on stack
;
;  Copy FlashEE page erase routine into RAM
;
  ldhx    #RamEraseSize       ; initialize pointer
EEErasel:

lda     RamEraseEE-1,x              ; get program from Flash
psha                                ; copy into Stack
dbnzx   EEErasel                    ; decrement pointer and loop back until done

; Execute program routine in RAM.
;
tsx                                 ; use H:X to point to RAM executable routine
tpa                                 ; get CCR
sei                                 ; disable all interrupts
jsr ,x                              ; erase Flash page

ais     #RamEraseSize               ; deallocate stack space used
pulh                                ; restore
pulx                                ; address pointer
tap                                 ; restore previous CCR
rts                                 ; return

;* Find First Erased FlashEE Location Subroutine  ==========================================
;* This subroutine is used to find the first erased FlashEE block, starting at the
;* address being pointed to by H:X. The FlashEE data block size is passed forward in ACC.
;* Calling convention:
;*
;* ldhx    #address
;* lda     #EE_BlockSize
;* jsr     EEFindFirst
;* Returns: H:X -> first erased FlashEE block address
;* ACC = number of erased FlashEE bytes left
;* CCRZ = set if erased location successfully found, otherwise clear
;* Changes: everything
;*
EEFindFirst:
psha                                ; save FlashEE data block size
lda     #EE_FlashRow                ; get Flash Program Row size
psha                                ; save on stack as a counter

EEFindFirst1:

lda     #EE_FlashErased             ; get erased Flash data
cmp ,x                              ; check if Flash location is erased
beq     EEFindFirst2                ; exit if so
pula                                ; else, get counter
sub 1,sp                            ; adjust by subtracting block size
bmi     EEFindFirst3                ; exit if out of room
psha                                ; else, save new count
tax                                ; perform
add 2,sp                            ; 16-bit addition
tax                                ; in order
pshh                                ; to advance
pula                                ; FlashEE pointer
adc #0                              ; forwards
psha                                ; to the
pulh                                ; next block
bra     EEFindFirst1                ; loop back
EEFindFirst2:
pula                                ; retrieve remaining erased bytes count
EEFindFirst3:
ais     #1                          ; deallocate stack usage
rts                                 ; return

/* FlashEE Block Program Subroutine  ==============================================================
/*
/* This subroutine will write the block of data being pointed to by H:X into the FlashEE.
/* 80 bytes of stack space is used, including this subroutine call return address. The
/* FlashEE data block size is passed forward in ACC.
/*
/* Calling convention:
/*
/* 1dhx    #EE_StartAddr
/* 1pshx
/* 1pshh
/* 1ldhx    #BlockSourceAddress
/* 1lda    #EE_BlockSize
/* 1jsr     WriteEE
/* 1ais     #2
/*
/* Returns:    nothing
/*
/* Changes:    everything
/*
EEWrite:
psha                                ; save FlashEE data block size
pshx                                ; save block
pshh                                ; source pointer

; Check for room within first FlashEE row.
; 1dx    7,sp                        ; get first FlashEE row address lsb
  lda    6,sp                        ; get first FlashEE
  psha                                ; row address
  pulp                                ; msb
  lda    3,sp                        ; get FlashEE data block size
  bsr     EEFindFirst                 ; go find next free location
  cmp     3,sp                        ; check if there’s room for another data block
  bpl     EEWrite1                    ; continue if so

; Check for room within second FlashEE row (which is within the same erase page).
; 1dx    7,sp                        ; get first FlashEE row address lsb
  lda    6,sp                        ; get first FlashEE
  psha                                ; row address
  pulp                                ; msb
  aix     #EE_FlashRow                ; H:X now points to second program row
  lda    3,sp                        ; get FlashEE data block size
  bsr     EEFindFirst                 ; go find next free location
  cmp     3,sp                        ; check if there’s room for another data block
  bpl     EEWrite1                    ; continue if so
; If there’s no room, then erase entire FlashEE page.

ldx 7,sp        ; get first FlashEE row address lsb
lda 6,sp        ; get first FlashEE row address
psha            ;   msb
pulh            ; erase entire FlashEE page

bsr EErase

EEWrite1:
pshx            ; save block
pshh            ; destination pointer

; Copy FlashEE byte program routine into RAM

ldhx #RamWriteSize       ; initialize program size counter

EEWrite2:

lda RamWriteEE-1,x      ; get program from Flash
psha                    ; copy onto stack
dbnzx EEWrite2          ; decrement pointer and loop back until done

; Prepare source and destination pointers and FlashEE block byte counter.

tsx            ; use H:X to point to RAM executable routine
lda (RamWriteSize+2),x ; get source address msb
sta (RamWriteSRC),x   ; save it in RAM executable routine
lda (RamWriteSize+3),x ; get source address lsb
sta (RamWriteSRC+1),x ; save it in RAM executable routine
lda (RamWriteSize),x  ; get destination address msb
sta (RamWriteDST1),x  ; save it in
sta (RamWriteDST2),x  ; RAM executable routine
lda (RamWriteSize+1),x ; get destination address lsb
sta (RamWriteDST1+1),x ; save it in
sta (RamWriteDST2+1),x ; RAM executable routine

; Execute program routine in RAM.

tpa            ; get CCR for current I-bit status
sei            ; disable all interrupts
jsr ,x         ; write data into Flash

ais #{RamWriteSize+5} ; deallocate stack space used
tap            ; restore previous CCR, specifically the I-bit
rts            ; return

;* RAM Executable FlashEE Block Program Subroutine ------------------------------------------
;* This subroutine controls the FlashEE block programming sequence.
;* This subroutine is copied into and executed from RAM. It is self-modifying and expects
;* to be called via "jsr ,x".
;*
RamWriteEE:
    psha                      ; save previous CCR on stack
;
; Step 1:
; Set PGM, read the Flash Block Protect Register and write any data to first Flash address.
;
    lda     #{PGM}                      ; set PGM control bit
    sta     flcr                        ; in Flash Control Register
    lda     flbpr                       ; read from Flash Block Protect Register
;
; "RamWriteDST1" is location offset relative to "RamWriteEE".
; This RAM location is used as a 16-bit destination address pointer.
;
    RamWriteDST1:    equ     {*-RamWriteEE+1}
    sta     $FFFF                       ; write any data to first Flash address
;
; Step 2:
; Wait for >10us, then set HVEN, then wait for >5us.
;
    lda     #RamDelay10                 ; wait
    dbnza   *                           ; for 10us
    lda     #{PGM | HVEN}               ; set HVEN control bit
    sta     flcr                        ; in Flash Control Register
    lda     #RamDelay5                  ; wait
    dbnza   *                           ; for 5us
;
; Step 3:
; Write data to Flash and wait for 30 - 40us. Repeat until done.
;
    RamWriteEE1:
    ; "RamWriteSRC" is location offset relative to "RamWriteEE".
    ; This RAM location is used as a 16-bit source address pointer.
;
    RamWriteSRC:    equ     {*-RamWriteEE+1}
    lda     $FFFF                       ; get data
;
; "RamWriteDST2" is location offset relative to "RamWriteEE".
; This RAM location is used as a 16-bit destination address pointer.
;
    RamWriteDST2:    equ     {*-RamWriteEE+1}
    sta     $FFFF                       ; write data to Flash
;
; Advance source and destination pointers.
; This sequence requires between 14 to 22 cycles.
;
    inc     {RamWriteDST2+1},x         ; advance the destination address lsb
    bne     RamWriteEE2                ; skip if no overflow
    inc     RamWriteDST2,x             ; else, advance the destination address msb

RamWriteEE2:
    inc     {RamWriteSRC+1},x           ; advance the source address lsb
    bne     RamWriteEE3                ; skip if no overflow
    inc     RamWriteSRC,x              ; else, advance the source address msb
RamWriteEE3:
  lda     #(RamDelay30-4)             ; wait
  dbnza   *                           ; for a total of 30us
  dbnz    (RamWriteSize+8),sp,RamWriteEE1 ; decrement byte counter, loop back til done
;
;  Step 4:
;  Clear PGM and wait for >5us.
;
  lda     #{HVEN}                     ; clear PGM control bit
  sta     flcr                        ; in Flash Control Register
  lda     #RamDelay5                  ; wait
  dbnza   *                           ; for 5us
;
;  Step 5:
;  Clear HVEN, wait >1us and return.
;
  clra                                ; clear HVEN control bit
  sta     flcr                        ; in Flash Control Register
  pula                             ; retrieve previous CCR
  brn     *                           ; wait for at least 1us before
  rts                              ; returning

RamWriteSize:   equ     {*-RamWriteEE}
eetest.asm

.header 'MC68HC908GP32 Flash as EEPROM Test'
.base 10t
.pagewidth 130
.page length 90

;********************************************************************************************
;*                                                                                          *
;*  Flash as EEPROM Test - MC68HC908GP32                                                   *
;*                                                          Copyright (c) Motorola, 2001    *
;*                                                                                          *
;********************************************************************************************
;*                                                                                          *
;*  Test program for FlashEE.                                                               *
;*                                                                                          *
;********************************************************************************************
;*                                                                                          *
;*  File name:          eetest.asm                      Current Release Level:      1.0     *
;*  Last Edit Date:     15-Jun-01                       Classification:             ES      *
;*  Include Files:      gp32.equ            : MC68HC908GP32 MCU definitions                 *
;*                      flashee.equ         : FlashEE parameters                            *
;*                      flashee.asm         : FlashEE routines                              *
;*  Assembler:          P&E’s CASM08Z                   Version:    3.16                    *
;*  Target:             MC68HC908GP32                                                       *
;*  Documentation:      MC68HC908GP32/H  Rev 3                                              *
;*                      Motorola Microcontroller Technical Data                             *
;*                                                                                          *
;********************************************************************************************
;*                                                                                          *
;*  Author:             DHJ Klotz                                                           *
;*  First Release:      15-Jun-01                                                           *
;*  Update History:                                                                         *
;*  Rev     Date       Author  Description of Change                                        *
;*  ------  ---------  ------  -----------------------------------------------------------  *
;*  ES 1.0  15-Jun-01  DHJK    Initial release.                                             *
;*                                                                                          *
;********************************************************************************************
;*                                                                                          *
;*  Notes:                                                                                  *
;*                                                                                          *
;********************************************************************************************
;*                                                                                          *
;*    Motorola reserves the right to make changes without further notice to any product   *
;*    herein to improve reliability, function, or design. Motorola does not assume any    *
;*    liability arising out of the application or use of any product, circuit, or software *
;*    described herein; neither does it 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 life, or for any other application in which the     *
;*    failure of the Motorola product could create a situation where personal injury or
.; death may occur. Should Buyer purchase or use Motorola products for any such
.; intended or unauthorized application, Buyer shall indemnify and hold Motorola and
.; its officers, employees, subsidiaries, affiliates, and distributors harmless against
.; all claims, costs, damages, and expenses, and reasonable attorney fees arising out
.; of, directly or indirectly, any claim of personal injury or death associated with
.; such unintended or unauthorized use, even if such claim alleges that Motorola was
.; negligent regarding the design or manufacture of the part.
.;
.; Motorola and the Motorola logo are registered trademarks of Motorola Ltd.
.;
.;********************************************************************************************
.
.set simulate ; enable simulation situational assembly

.; Microcontroller Peripheral Equates  ****************************************************
.;
.; nolist
.include "gp32.equ" ; include microcontroller definitions file
;
;list

init_config2: equ %00000001 ; initial Configuration Register 2
init_config1: equ %00000001 ; initial Configuration Register 1
init_stack: equ ram_last ; initialize stack pointer to last RAM location

;;; Serial Communications Interface (SCI)

init_scc1: equ %01000000 ; initial SCI Control Register 1
init_scc2: equ %00001100 ; initial SCI Control Register 2
init_scc3: equ %00000000 ; initial SCI Control Register 3
init_scbr: equ %00000000 ; initial SCI Baud Rate Register

.include "flashee.equ" ; include FlashEE parameters file

.; Global Variables  **************************************************************************
.;
.; org ram_start
buffer: ds 64 ; generic input data buffer

.; Power-on Reset  **************************************************************************
.;
.; org $9000

Start:
sta copctl ; clear the COP counter
mov #init_config2,config2 ; initialize Configuration Register 2
mov #init_config1,config1 ; initialize Configuration Register 1
ldhx #init_stack+1 ; initialize
txs ; the stack pointer

; Initialize the CGM for 7.3728 MHz bus speed from 32.768 kHz crystal.

ldhx #bus7372800 ; point to 7.372800 MHz parameters
jsr PLLset ; change bus speed

AN2183
Clear all RAM.

ClearRAM:

```
ldhx #ram_start                  ; point to start of RAM
clr     ,x                          ; clear RAM location
aix     #1                          ; advance pointer
cphx    #ram_last+1                 ; done ?
      bne ClearRAM                    ; loop back if not
```

Initialize Port I/O and Variables

```
mov     #init_scc1,scc1             ; initialize SCI Control Register 1
mov     #init_scc2,scc2             ; initialize SCI Control Register 2
mov     #init_scc3,scc3             ; initialize SCI Control Register 3
mov     #init_scbr,scbr             ; initialize SCI Baud Rate Register
cli                                 ; enable all interrupts
```

Main Control Loop

```
; Interface commands:
; 'W' = write following string into FlashEE
; 'R' = read back current FlashEE block data
; 'D' = dump entire FlashEE
; 'E' = erase entire FlashEE
; cmd_read: equ     'R'                  ; Read command
cmd_erase: equ     'E'                  ; Erase command
cmd_write: equ     'W'                  ; Write command
cmd_dump: equ     'D'                  ; Dump command

main:

```
ldhx #msg_hello                  ; point to hello message
jsr     PrintString                 ; output it
jsr     GetChar                     ; get a character from the SCI
cmp     #ascii_CR                   ; check for ASCII carriage return
      beq     main                        ; just loop back if so
jsr     PutChar                     ; else, echo character back
      and     #$DF                        ; convert to uppercase
```

Check for Read command and execute.

```
Check_Read:

cmp     #cmd_read               ; check for Read command
      bne Check_Erase              ; skip if not
```

```
jsr     GetChar                     ; get a character from the SCI
jsr     PutChar                     ; echo character back
cmp     #1                   ; check if target is 1st FlashEE
      beq     Check_Read1              ; continue if so
cmp     #2                   ; check if target is 2nd FlashEE
      beq     Check_Read2              ; continue if so
      jmp     Check_What              ; else, respond to unknown command
```
Check_Read1:
    ldhx #EE_StartAddr1              ; point to start of 1st FlashEE
    lda #EE_BlockSize1              ; get 1st FlashEE data block size
    bra Check_Read3                 ; continue

Check_Read2:
    ldhx #EE_StartAddr2              ; point to start of 2nd FlashEE
    lda #EE_BlockSize2              ; get 2nd FlashEE data block size

Check_Read3:
    psha                                ; save FlashEE data block size on stack
    jsr EERead                      ; move pointer to FlashEE data
    cmp #EE_FlashErased             ; check if data is erased
    bne Check_Read4                 ; skip if not
    pula                                ; else, deallocate stack usage
    ldhx #msg_erased                 ; point to FlashEE erased message
    jsr PrintString                 ; output it
    bra main                        ; loop back to top

Check_Read4:
    lda # '  '                         ; output a space
    jsr PutChar                     ;  a space
    lda ,x                          ; read FlashEE data
    jsr PutHexByte                  ; output it
    aix #1                          ; advance FlashEE pointer
    dbnz 1,sp,Check_Read4            ; loop back until done
    pula                                ; deallocate stack usage
    bra main                        ; loop back to top

Check_Erase:
    cmp #cmd_erase                  ; check for Erase command
    bne Check_Write                 ; skip if not
    jsr GetChar                     ; get a character from the SCI
    jsr PutChar                     ; echo character back
    cmp #1                       ; check if target is 1st FlashEE
    beq Check_Erase1                ; continue if so
    cmp #2                       ; check if target is 2nd FlashEE
    beq Check_Erase2                ; continue if so
    jmp Check_What                  ; else, respond to unknown command

Check_Erase1:
    ldhx #EE_StartAddr1              ; point to start of 1st FlashEE
    bra Check_Erase3                 ; continue

Check_Erase2:
    ldhx #EE_StartAddr2              ; point to start of 2nd FlashEE

Check_Erase3:
    jsr EEErase                     ; Erase FlashEE
    ldhx #msg_erased                 ; point to FlashEE erased message
    jsr PrintString                 ; output it
    bra main                        ; loop back to top

; Check for Write command and execute.  ===================================================
Check_Write:
  cmp     #cmd_write                  ; check for Write command
  bne     Check_Dump                  ; skip if not

;  jsr     GetChar                     ; get a character from the SCI
  jsr     PutChar                     ; echo character back
  cmp     #1'                         ; check if target is 1st FlashEE
  beq     Check_Write1                ; continue if so
  cmp     #2'                         ; check if target is 2nd FlashEE
  beq     Check_Write2                ; continue if so
  jmp     Check_What                  ; else, respond to unknown command

Check_Write1:
  ldhx    #EE_StartAddr1              ; point to start of 1st FlashEE
  lda     #EE_BlockSize1              ; get 1st FlashEE data block size
  bra     Check_Write3                ; continue

Check_Write2:
  ldhx    #EE_StartAddr2              ; point to start of 2nd FlashEE
  lda     #EE_BlockSize2              ; get 2nd FlashEE data block size

Check_Write3:
  pshx                                ; save FlashEE address lsb temporarily
  pshh                                ; save FlashEE address msb temporarily
  psha                                ; save FlashEE data block size temporarily
  lda     # '='                         ; output '=' to indicate
  jsr     PutChar                     ; ready for data

ldhx    #buffer                     ; reset buffer pointer

Check_Write4:
  jsr     GetHexByte                  ; go retrieve data byte
  bne     Check_What                  ; output error if not hexadecimal
  sta     ,x                          ; save data in buffer
  lda     # ' '                         ; output
  jsr     PutChar                     ; a space
  aix     #1                          ; advance buffer pointer
  dbnz    1,sp,Check_Write4           ; loop back until entire FlashEE block received
  pula                                ; deallocate stack usage

;  pula                                ; retrieve FlashEE data block size
  ldhx    #buffer                     ; point to buffer (FlashEE address is on stack)
  jsr     EEWrite                     ; write data into FlashEE
  ais     #2                          ; deallocate stack usage
  jmp     main                        ; loop back to top

;  Check for Dump command and execute.

Check_Dump:
  cmp     #cmd_dump                  ; check for Dump command
  bne     Check_What                  ; skip if not

;  jsr     GetChar                     ; get a character from the SCI
  jsr     PutChar                     ; echo character back
  cmp     #1'                         ; check if target is 1st FlashEE
  beq     Check_Dump1                ; continue if so
  cmp     #2'                         ; check if target is 2nd FlashEE
  beq     Check_Dump2                ; continue if so
  jmp     Check_What                  ; else, respond to unknown command
Check_Dump1:
    ldhx    #EE_StartAddr1              ; point to start of 1st FlashEE
    bra     Check_Dump3                 ; continue

Check_Dump2:
    ldhx    #EE_StartAddr2              ; point to start of 2nd FlashEE

Check_Dump3:
    lda     #8                          ; initialize
    psha                                ; line counter

    txa                                 ; perform
    sub     #16                         ; 16-bit subtraction
    psha                                ; in order to adjust
    pshh                                ; FlashEE pointer
    puls                                ; backwards one dump
    sbc     #0                          ; print row
    psha                                ; and put pointer on the stack

Check_Dump4:
    ldhx    #msg_CRLF                   ; point to <CR><LF> message
    jsr     PrintString                 ; output it
    pulh                                ; restore
    pulx                                ; location pointer
    aix     #16                         ; add offset
    pshx                                ; save result
    pshh                                ; back on stack
    lda     #16                         ; initialize
    psha                                ; byte counter

Check_Dump5:
    lda     ','                        ; output
    jsr     PutChar                     ; a space
    lda     ',x                       ; read FlashEE data
    aix     #1                         ; move location pointer
    jsr     PutHexByte                  ; output it
    dbnz    1,sp,Check_Dump5            ; loop back until all bytes done
    pula                                ; deallocate stack usage
    dbnz    3,sp,Check_Dump4            ; loop back until all lines done
    ais     #3                          ; deallocate stack usage
    jmp     main                        ; loop back to top

; Handle unknown commands.

Check_What:
    ldhx    #msg_what                   ; point to unknown command message
    jsr     PrintString                 ; output it
    jmp     main                        ; loop back to top

; Messages
===================================================================================
; ascii_CR:       equ     $0D                 ; ASCII carriage return
ascii_LF:       equ     $0A                 ; ASCII line feed
; msg_hello:      db      ascii_CR,ascii_LF, 'FlashEE>',0
msg_erased:     db      'FlashEE erased',0
msg_what:       db      '- what?',0
msg_CRLF:       db      ascii_CR,ascii_LF,0
/* PrintString Subroutine */
/* This subroutine will output the null terminated string pointed to by H:X to the SCI. */
/* Calling convention: */
*/
ldhx #string ; point to start of string
jsr PrintString ; go output it
*/
>Returns: nothing
*/
Changes: H:X
*/
PrintString1:
brclr SCTE,scs1,PrintString1 ; wait until SCI transmitter is empty
mov x+,scdr ; output character to the SCI and advance pointer
PrintString:
tst ,x ; test string character
bne PrintString1 ; loop back if not null
rts ; else, return

/* PutChar Subroutine */
/* This subroutine will output the character passed in ACC to the SCI. */
/* C function prototype: */
/* void PutChar (char data); */
/* Calling convention: */
*/
lda data ; get character
jsr PutChar ; go output it
*/
>Returns: nothing
*/
Changes: nothing
*/
PutChar:
brclr SCTE,scs1,PutChar ; wait until SCI transmitter is empty
sta scdr ; output character to the SCI
rts ; return

/* GetChar Subroutine */
/* This subroutine will wait forever for a character to be received by the SCI and then */
/* returns with that character in ACC. No error checking is performed. Note that this */
/* is the primary loop where the COP counter is cleared. */
/* C function prototype: */
/* char GetChar (void); */
/* Calling convention: */
*/
jsr GetChar ; get a character from the SCI
GetChar:
    sta copctl ; clear the COP counter
    brclr SCRF,scs1,GetChar ; wait forever until SCI receiver is full
    lda scdr ; get data
    rts ; return

PutHexByte:
    psha ; save ACC temporarily
    nsa ; move upper nibble down
    bsr FromHex ; convert it to ASCII
    bsr PutChar ; output it
    pula ; retrieve data
    bsr FromHex ; convert lower nibble to ASCII
    bsr PutChar ; output it
    rts ; return

GetHexByte:
    bsr GetChar ; get msb character from the SCI
    bsr PutChar ; echo it back
    bsr IsHex ; check if valid ASCII hex character
    bne GetHexByte2 ; exit if not
    bsr ToHex ; convert ASCII hex character to hex value
    nsa ; swap lower nibble up
    psha ; save temporarily

PutChar:

IsHex:

ToHex:

FromHex:

GetChar:

GetHexByte2:

AN2183

For More Information On This Product, Go to: www.freescale.com
jsr GetChar  ; get lsb character from the SCI
bsr PutChar  ; echo it back
bsr IsHex    ; check if valid ASCII hex character
bne GetHexByte1 ; exit if not
bsr ToHex    ; convert ASCII hex character to hex value
add 1,sp     ; combine msb and lsb nibbles
bit #0        ; CCRZ= 1
GetHexByte1:  
ais #1        ; deallocate local variable
GetHexByte2:  
  rts          ; return

;* FromHex Subroutine  ===================================================================
;* This subroutine converts the value passed in the lower nibble of ACC to its ASCII equivalent.
;* Calling convention:
;* jsr FromHex
;* Returns: ACC= data.
;* Changes: ACC
FromHex:
  and #$0F      ; mask off upper nibble
  add #'0'      ; add ASCII offset for '0'
  cmp #'9'      ; check if result is between '0' to '9'
  bls FromHex1  ; skip if so
  add #7        ; else, adjust for value between 'A' to 'F'
FromHex1:
  rts          ; return

;* ToHex Subroutine  =====================================================================
;* This subroutine converts the ASCII hex value passed in ACC to a binary hex value.
;* Calling convention:
;* lda data
;* jsr ToHex
;* Returns: ACC= data.
;* Changes: ACC
ToHex:
  sub #'0'      ; adjust first by subtracting '0'
  cmp #9        ; check if value was between '0' to '9'
  bls ToHex1    ; exit if so
  sub #7        ; else, adjust for value between 'A' to 'F'
ToHex1:
  rts          ; return
;* IsHex Subroutine  ================================================================
;* This subroutine checks if the value passed in ACC is a valid ASCII hex character within
;* the ranges of '0' to '9' or 'A' to 'F' or 'a' to 'f'. Adjusts ACC if lowercase.
;* Calling convention:
;*      lda     data
;*      jsr     IsHex
;* Returns:    CCRZ= 1 if data is a valid hex character. Otherwise, CCRZ= 0.
;* Changes:    ACC (if lowercase)
;*
IsHex:
    cmp     # '0'                         ; check value against '0'
    blo     IsntHex                     ; not hex if lower
    cmp     # '9'                         ; check value against '9'
    bls     IsHex1                      ; is hex if lower
    cmp     # 'A'                         ; check value against 'A'
    blo     IsntHex                     ; not hex if lower
    cmp     # 'F'                         ; check value against 'F'
    bls     IsHex1                      ; is hex if lower
    sub     #$20                        ; adjust to uppercase
    cmp     # 'A'                         ; check value against 'A'
    blo     IsntHex                     ; not hex if lower
    cmp     # 'F'                         ; check value against 'F'
    bhi     IsntHex                     ; isnt hex if higher
    jsr     IsntHex                     ; return

IsHex1:
    bit     #0                          ; CCRZ= 1

IsntHex:
    rts                                 ; return

;* CGM PLL Bus Frequency Change Subroutine  ========================================
;* This subroutine will program the CGM PLL to change the bus frequency in accordance with
;* the data being pointed to by H:X.
;* Calling convention:
;*      ldhx    #busfreq_table
;*      jsr     PLLset
;* Returns:    no data
;* Changes:    H:X
;*
PLLset:
    bclr    BCS,pctl                    ; select external reference as base clock
    bclr    PLLON,pctl                  ; turn off PLL
    mov     x+,pctl1                     ; program P & E
    mov     x+,pmrs                      ; program L
    mov     x+,pmsh                      ; program N msb
    mov     x+,pmsl                      ; program N lsb
    bset    AUTO,pbwc                    ; enable automatic bandwidth control
    bset    PLLON,pctl1                  ; turn on PLL
PLLwait:
    .ifnot simulate
        brclr LOCK,pbwc,PLLwait  ; wait for PLL to lock
    .endif
    bset BCS,pctl  ; select VCO as base clock
    rts  ; return

;* 8.003584 MHz bus frequency parameters
;*
bus8003584:
    db $02  ; P & E
    db $D0  ; L
    db $03  ; N msb
    db $D1  ; N lsb
    db $01  ; delay_msb
    db $F4  ; delay_lsb
    db %00110000  ; SCI Baud Rate Register = 9600
    db 200  ; LCD 100us delay parameter

;* 7.3728 MHz bus frequency parameters
;*
bus7372800:
    db $02  ; P & E
    db $C0  ; L
    db $03  ; N msb
    db $84  ; N lsb
    db $01  ; delay_msb
    db $CC  ; delay_lsb
    db %00010010  ; SCI Baud Rate Register = 9600
    db 184  ; LCD 100us delay parameter

    include "flashee.asm"  ; include FlashEE routines

;* Dummy Interrupt Vector Handler ****************************
;
Dummy:
    rti

;* Vectors  **************************************************************
;*
    org vec_timebase  ; Timebase vector
    dw Dummy
    org vec_adc  ; ADC vector
    dw Dummy
    org vec_kbd  ; Keyboard vector
    dw Dummy
    org vec_scitx  ; SCI transmit vector
    dw Dummy
    org vec_scirx  ; SCI receive vector
    dw Dummy
    org veciscierr  ; SCI error vector
    dw Dummy
    org vec_spitx  ; SPI transmit vector
    dw Dummy
    org vec_spirx  ; SPI receive vector
    dw Dummy
    org vec_tim2ov  ; Timer 2 overflow vector
    dw Dummy
org     vec_tim2ch1                 ; Timer 2 channel 1 vector
dw      Dummy

org     vec_tim2ch0                 ; Timer 2 channel 0 vector
dw      Dummy

org     vec_tim1ov                  ; Timer 1 overflow vector
dw      Dummy

org     vec_tim1ch1                 ; Timer 1 channel 1 vector
dw      Dummy

org     vec_tim1ch0                 ; Timer 1 channel 0 vector
dw      Dummy

org     vec_pll                     ; PLL vector
dw      Dummy

org     vec_irq                     ; IRQ vector
dw      Dummy

org     vec_swi                     ; SWI vector
dw      Dummy

org     vec_reset                   ; Reset vector
dw      Start

dw      Start
gp32.equ

;**********************************************************************************************

;*      MC68HC908GP32 Definitions

;*      Copyright (c) Derrick HJ Klotz, 2001

;*      File name:          gp32.equ                    Current Release Level:      1.0
;*      Last Edit Date:     22-Feb-00                   Classification:             ES
;*      Include Files:      none                                             Version:    3.06
;*      Assembler:          P&E's CASM08
;*      Target Device:      MC68HC908GP32
;*      Documentation:      MC68HC908GP32/H Rev 3 Microcontroller Technical Data

;*      Author:             DHJ Klotz                   Location:   TOR
;*      First Release:      22-Feb-00

;*      Update History:
;*      Rev     Date        Author  Description of Change
;*      ------  ---------   ------  ----------------------------------------------------------
;*      ES 1.0  22-Feb-00   DHJK   Initial release.

;**********************************************************************************************

Motorola reserves the right to make changes without further notice to any product
herein to improve reliability, function, or design. Motorola does not assume any
liability arising out of the application or use of any product, circuit, or software
described herein; neither does it 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 life, or for any other application in which the
failure of the Motorola product could create a situation where personal injury or
death may occur. Should Buyer purchase or use Motorola products for any such
intended or unauthorized application, Buyer shall indemnify and hold Motorola and
its officers, employees, subsidiaries, affiliates, and distributors harmless against
all claims, costs, damages, and expenses, and reasonable attorney fees arising out
of, directly or indirectly, any claim of personal injury or death associated with
such unintended or unauthorized use, even if such claim alleges that Motorola was
negligent regarding the design or manufacture of the part.

Motorola and the Motorola logo are registered trademarks of Motorola Ltd.

AN2183
;; Memory Map and Interrupt Vectors
;*******************************************************************************************
; variables
;******************************************************************************
ram_start: equ $0040           ; start of RAM
ram_last: equ $023F           ; last RAM location
rom_start: equ $8000          ; start of ROM
rom_last: equ $FDFF           ; last ROM location
vec_timebase: equ $FFDC       ; Timebase vector
vec_adc: equ $FFDE            ; ADC vector
vec_kbd: equ $FFE0            ; Keyboard vector
vec_scitx: equ $FFE2          ; SCI transmit vector
vec_scirx: equ $FFE4          ; SCI receive vector
vec_scierr: equ $FFE6         ; SCI error vector
vec_spitx: equ $FFE8          ; SPI transmit vector
vec_spirx: equ $FFEA          ; SPI receive vector
vec_tim2ov: equ $FFEC         ; Timer 2 overflow vector
vec_tim2ch1: equ $FFEE         ; Timer 2 channel 1 vector
vec_tim2ch0: equ $FFF0         ; Timer 2 channel 0 vector
vec_tim1ov: equ $FFF2         ; Timer 1 overflow vector
vec_tim1ch1: equ $FFF4         ; Timer 1 channel 1 vector
vec_tim1ch0: equ $FFF6         ; Timer 1 channel 0 vector
vec_pll: equ $FFF8            ; PLL vector
vec_irq: equ $FFFA            ; IRQ vector
vec_swi: equ $FFFC            ; SWI vector
vec_reset: equ $FFFFFF         ; Reset vector

;; Input/Output (I/O) Ports
;*******************************************************************************************
; variables
;******************************************************************************
porta: equ $00            ; Port A Data Register
portb: equ $01            ; Port B Data Register
portc: equ $02            ; Port C Data Register
portd: equ $03            ; Port D Data Register
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
porte: equ $08            ; Port E Data Register
ddre: equ $0C             ; Port E Data Direction Register
ptapue: equ $0D           ; Port A Input Pullup Enable Register
ptcpue: equ $0E           ; Port C Input Pullup Enable Register
ptdpue: equ $0F           ; Port D Input Pullup Enable Register

;; Serial Peripheral Interface Module (SPI)
;*******************************************************************************************
; variables
;******************************************************************************
spcr: equ $10             ; SPI Control Register
SPRIE: equ 7              ; SPI receiver interrupt enable bit
SPMSTR: equ 5             ; SPI master bit
CPOL: equ 4               ; clock polarity bit
CPHA: equ 3               ; clock phase bit
SPWOM: equ 2              ; SPI wired-or mode bit
SPE: equ 1                ; SPI enable
SPTIE: equ 0              ; SPI transmit interrupt enable
Application Note

**SPI Status and Control Register**

- **SPSCR**: equ $11
- **SPRF**: equ 7
- **ERRIE**: equ 6
- **OVRF**: equ 5
- **MODF**: equ 4
- **SPTE**: equ 3
- **SPR1**: equ 1
- **SPR0**: equ 0

**SPI Data Register**

- **SPDR**: equ $12

---

**Serial Communications Interface (SCI)**

- **SCC1**: equ $13
- **LOOPS**: equ 7
- **ENSCI**: equ 6
- **TXINV**: equ 5
- **M**: equ 4
- **WAKE**: equ 3
- **ILTY**: equ 2
- **PEN**: equ 1
- **PTY**: equ 0

- **SCC2**: equ $14
- **SCTIE**: equ 7
- **TCIE**: equ 6
- **SCRIIE**: equ 5
- **ILIE**: equ 4
- **TE**: equ 3
- **RE**: equ 2
- **RWU**: equ 1
- **SBK**: equ 0

- **SCC3**: equ $15
- **R8**: equ 7
- **T8**: equ 6
- **DMARE**: equ 5
- **DMATE**: equ 4
- **ORIE**: equ 3
- **NEIE**: equ 2
- **FEIE**: equ 1
- **PEIE**: equ 0

---

For More Information On This Product,
Go to: www.freescale.com
scs2:  equ  $17 ; SCI Status Register 2
BKF:   equ  1    ; break flag bit
RPF:   equ  0    ; reception in progress flag bit
;
scdr:  equ  $18 ; SCI Data Register
scbr:  equ  $19 ; SCI Baud Rate Register

;* Keyboard Interrupt Module (KBI)
******************************************************************************
;*
intkbscr:  equ  $1A    ; Keyboard Status and Control Register
KEYF:      equ  3    ; keyboard flag bit
ACKK:      equ  2    ; keyboard acknowledge bit
IMASKK:    equ  1    ; keyboard interrupt mask bit
MODEK:     equ  0    ; keyboard triggering sensitivity bit
;
intkbier:  equ  $1B    ; Keyboard Interrupt Enable Register
KBIE7:     equ  7
KBIE6:     equ  6
KBIE5:     equ  5
KBIE4:     equ  4
KBIE3:     equ  3
KBIE2:     equ  2
KBIE1:     equ  1
KBIE0:     equ  0

;* Timebase Module (TBM)
******************************************************************************
;*
tbcr:      equ  $1C    ; Timebase Control Register
TBIF:      equ  7    ; timebase interrupt flag
TBR2:      equ  6    ; \
TBR1:      equ  5    ; timebase rate selection
TBR0:      equ  4    ; /
TACK:      equ  3    ; timebase acknowledge
TBIE:      equ  2    ; timebase interrupt enable
TBON:      equ  1    ; timebase enabled

;* External Interrupt (IRQ)
******************************************************************************
;*
tintscr:    equ  $1D    ; IRQ Status and Control Register
IRQF:       equ  3    ; IRQ flag bit
ACK:        equ  2    ; IRQ interrupt request acknowledge bit
IMASK:      equ  1    ; IRQ interrupt mask bit
MODE:       equ  0    ; IRQ edge/level select bit

;* Configuration Registers (CONFIG)
******************************************************************************
;*
config2:    equ  $1E    ; Configuration Register 2
config1:    equ  $1F    ; Configuration Register 1
"* Timer Interface module (TIM) 
***********************************************************************
;*  
;*  t1sc:  equ $20 ; Timer 1 Status and Control Register
;*  t2sc:  equ $2B ; Timer 2 Status and Control Register
;*  TOF:    equ 7 ; TIM overflow flag bit
;*  TOIE:   equ 6 ; TIM overflow interrupt enable bit
;*  TSTOP:  equ 5 ; TIM stop bit
;*  TRST:   equ 4 ; TIM reset bit
;*  PS2:    equ 2 ; prescaler select bits
;*  PS1:    equ 1 ; 
;*  PS0:    equ 0 ; / 
;*  t1sc0:  equ $25 ; Timer 1 Channel 0 Status and Control Register
;*  t1sc1:  equ $28 ; Timer 1 Channel 1 Status and Control Register
;*  t2sc0:  equ $30 ; Timer 2 Channel 0 Status and Control Register
;*  t2sc1:  equ $33 ; Timer 2 Channel 1 Status and Control Register
;*  CHXF:   equ 7 ; channel x flag bit
;*  CHxIE:  equ 6 ; channel x interrupt enable
;*  MSxB:   equ 5 ; channel x mode select bit B
;*  MSxA:   equ 4 ; channel x mode select bit A
;*  ELSxB:  equ 3 ; channel x edge/level select bit B
;*  ELSxA:  equ 2 ; channel x edge/level select bit A
;*  TOVx:   equ 1 ; channel x toggle on overflow bit
;*  CHxMAX: equ 0 ; channel x maximum duty cycle bit
;*  t1cnt:  equ $21 ; Timer 1 Counter Register
;*  t1mod:  equ $23 ; Timer 1 Counter Modulo Register
;*  t1ch0:  equ $26 ; Timer 1 Channel 0 Register
;*  t1ch1:  equ $29 ; Timer 1 Channel 1 Register
;*  t2cnt:  equ $2C ; Timer 2 Counter Register
;*  t2mod:  equ $2E ; Timer 2 Counter Modulo Register
;*  t2ch0:  equ $31 ; Timer 2 Channel 0 Register
;*  t2ch1:  equ $34 ; Timer 2 Channel 1 Register

;*  Clock Generator Module (CGMC) 
***********************************************************************
;*  
;*  pctl:      equ $36 ; PLL Control Register
;*  PLLIE:     equ 7 ; PLL interrupt enable bit
;*  PLLF:      equ 6 ; PLL interrupt flag bit
;*  PLLON:     equ 5 ; PLL on bit
;*  BCS:       equ 4 ; base clock select bit
;*  PRE1:      equ 3 ; prescaler
;*  PRE0:      equ 2 ; program bits
;*  VPR1:      equ 1 ; VCO power-of-two
;*  VPR0:      equ 0 ; range select bits
;*  pbwc:      equ $37 ; PLL Bandwidth Control Register
;*  AUTO:      equ 7 ; automatic bandwidth control bit
;*  LOCK:      equ 6 ; lock indicator bit
;*  ACQ:       equ 5 ; acquisition mode bit
;*  pmsh:      equ $38 ; PLL Multiplier Select High Register
;*  pmsl:      equ $39 ; PLL Multiplier Select Low Register
;*  pmrs:      equ $3A ; PLL VCO Select Range Register
;*  pmds:      equ $3B ; PLL Reference Divider Select Register
/* Analog-to-Digital Converter (ADC)
*******************************************************************************************/

; adscr: equ $3C ; ADC Status and Control Register
COCO: equ 7 ; conversions complete flag
AIEL: equ 6 ; AIC int enable bit
ADCO: equ 5 ; AIC continuous conversion bit
ADCH4: equ 4 ; 
ADCH3: equ 3 ; 
ADCH2: equ 2 ; AIC channel select bits
ADCH1: equ 1 ; /
ADCH0: equ 0 ; /
;
ad: equ $3D ; ADC Data Register
;
adclk: equ $3E ; ADC Clock Register
ADIV2: equ 7 ; 
ADIV1: equ 6 ; ADC clock prescaler bits
ADIV0: equ 5 ; /
ADICLK: equ 4 ; ADC input clock select bit

/* System Integration Module (SIM)
*******************************************************************************************/

; sbsr: equ $FE00 ; SIM Break Status Register
SBSW: equ 1 ; SIM break stop/wait
;
srsr: equ $FE01 ; SIM Reset Status Register
POR: equ 7 ; power-on reset bit
PIN: equ 6 ; external reset bit
COP: equ 5 ; COP reset bit
ILOP: equ 4 ; illegal opcode reset bit
ILAD: equ 3 ; illegal opcode address reset bit
MODRST: equ 2 ; monitor mode entry module reset bit
LVI: equ 1 ; LVI reset bit
;
surb: equ $FE02 ; SIM Upper Byte Address Register
sbfcfr: equ $FE03 ; SIM Break Flag Control Register
BCFE: equ 7 ; break clear flag enable bit
int1: equ $FE04 ; Interrupt Status Register 1
int2: equ $FE05 ; Interrupt Status Register 2
int3: equ $FE06 ; Interrupt Status Register 3

/* Flash Memory
*******************************************************************************************/

; flcr: equ $FE08 ; Flash Control Register
HVEN: equ %00001000 ; high-voltage enable bit mask
MASS: equ %00000100 ; mass erase control bit mask
ERASE: equ %00000010 ; erase control bit mask
PGM: equ %00000001 ; program control bit mask
;
flbpr: equ $FF7E ; Flash Block Protect Register
Application Note

;* Breakpoint Module (BRK)
********************************************************************
;*
brkh:   equ     $FE09               ; Break Address Register High
brkl:   equ     $FE0A               ; Break Address Register Low
brkscr: equ     $FE0B               ; Break Status and Control Register
BRKE:   equ     7                       ; break enable bit
BRKA:   equ     6                       ; break active bit

;* Low-Voltage Inhibit (LVI)
********************************************************************
;*
lvisr:  equ     $FE0C               ; LVI Status Register
LVIOUT: equ     7                       ; LVI output bit

;* Computer Operating Properly (COP)
********************************************************************
;*
copctl: equ     $FFFF               ; COP Control Register

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.