This application note describes a method of in-circuit programming of FLASH memory via the Universal Serial Bus for the MC68HC908JB8.

For detailed specification on MC68HC908JB8 device, please refer to the data sheet; Freescale order number: MC68HC908JB8/D.

INTRODUCTION

The Freescale MC68HC908JB8 (hereafter referred as JB8) is a member of the HC08 Family of microcontrollers (MCUs). The features of the JB8 include a Universal Serial Bus (USB) interface, which makes this MCU suited for personal computer Human Interface Devices (HID), such as mice and keyboards.

On the JB8, 8k-bytes of FLASH memory is allocated for the user code, with an additional 16-bytes for user defined reset and interrupt vectors. A high voltage supply is not required by the JB8 for FLASH program or erase operations; as it is generated by an internal charge-pump.

In-circuit programming (ICP) is a process by which the device is programmed or erased with the device on the final circuit board — the target system. This allows the user code to be changed without having to remove the device off the target system for reprogramming; simplifying user code changes during product development, last minute changes during production, and code upgrades after the product is sold.

The following sections in this application note describes a method of implementing ICP using the USB as the communication link between host (PC) and HID.
OVERVIEW AND MEMORY USAGE

To use the USB interface as a communications link for ICP, the user code in the JB8 must be modified to recognize some pre-defined USB commands for ICP. Since the FLASH memory cannot be erased by code running in the same area as it is being erased, the code must be loaded into RAM and executed from RAM. The RAM size of 256-bytes in the JB8 is limited for the ICP scheme described. Therefore, the following ICP method uses code that is pre-programmed in an area of the JB8 memory. The user code is programmed to the remainder of the FLASH memory and block erase routines are used to erase the user code.

The JB8 must be initially programming with the ICP code in place, before it is soldered onto the printed circuit board. Figure 1 shows the FLASH memory usage for the JB8 ICP scheme.

From Figure 1, the user block ranges from $DC00 to $FBFF, and the user vectors block ranges from $FFF0 to $FFFF.

For this ICP scheme, the ICP code, from $F800 to $FBFF; and the user FLASH vectors do not get reprogrammed in an ICP operation. These two blocks are programmed before the JB8 is soldered onto the PCB. An ICP operation erases and programs the FLASH memory from $DC00 to $F7FF (the shaded area shown in Figure 1).
Vector Redirecting

Since the ICP scheme erases and reprograms the user code only, mass erase operation cannot be used. This means the user code is erased using multiple block erase operations. And because mass erase is not used, the user FLASH vectors cannot be erased during ICP (a fail-save mechanism allows only mass erase operation to erase the user FLASH vectors).

Since the user FLASH vectors are now fixed, these must be re-directed to the proper addresses for the interrupt service subroutines in the user code. This is achieved using “pseudo” vectors, which are 3-byte vectors containing a JMP instruction and the absolute address to the actual interrupt service subroutines in the user program. Figure 2 shows how the vectors are re-directed. The only vector that is not re-directed is the reset vector. The reset vector always points to $F800 — the start of the ICP code.

Table 1 lists interrupt vector addresses and the pseudo vector addresses for redirecting.

![Figure 2. Vector Redirecting](image-url)
Security Against Unauthorized Access

The contents of the 8 bytes, $FFF6 to $FFFD, are used as a passcode for entry into JB8’s monitor mode, where the monitoring software can have full access of the device FLASH memory, and thus allowing code dumps. Normally, this 8-byte passcode is virtually impossible to guess, as the starting address of these interrupt service routines are buried inside the user code.

If all eight pseudo vectors were fixed locations, say in an array from $F7E6 to $F7FD (3 bytes each), it would be quite easy to guess the 8-byte passcode. One way to make the guessing harder is to alter the sequence of the pseudo vectors in the array. The guessing is made even harder by shifting the array by one or two addresses, or by inserting blank slots in the array. The entire array can even be anywhere within the user code. The scheme implemented here is by embedding the critical 8 bytes randomly in the user code (the addresses Aw, Ax, Ay, and Az in Table 1).

Protection Against Power Failure During ICP

The ICP scheme must be designed to take into account of possible power failure during an ICP routine in progress. The command handler must be able to recover and complete the ICP routine. The ICP_FLAG word used for this purpose.

The ICP_FLAG

After reset, the ICP_FLAG word is read to determine whether the JB8 should enter normal operating mode or ICP mode. This word is at $F7FE, and is at the last two bytes in the user code area. This use of the ICP_FLAG is explained in the subsequent sections.

Table 1. Vector Addresses

<table>
<thead>
<tr>
<th>Vector Address</th>
<th>Pseudo Vector Address</th>
<th>Interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FFF0 : $FFF1</td>
<td>$F7F3 : $F7F4</td>
<td>Keyboard Interrupt</td>
</tr>
<tr>
<td>$FFF2 : $FFF3</td>
<td>$F7F6 : $F7F7</td>
<td>TIM Overflow</td>
</tr>
<tr>
<td>$FFF4 : $FFF5</td>
<td>$F7F9 : $F7FA</td>
<td>TIM Channel 1</td>
</tr>
<tr>
<td>$FFF6 : $FFF7</td>
<td>Aw : Aw+1 (1)</td>
<td>TIM Channel 0</td>
</tr>
<tr>
<td>$FFF8 : $FFF9</td>
<td>Ax : Ax+1 (1)</td>
<td>IRQ</td>
</tr>
<tr>
<td>$FFFA : $FFFB</td>
<td>Ay : Ay+1 (1)</td>
<td>USB</td>
</tr>
<tr>
<td>$FFFC : $FFFD</td>
<td>Az : Az+1 (1)</td>
<td>SWI</td>
</tr>
<tr>
<td>$FFFE : $FFFF</td>
<td>$F7FC : $F7FD</td>
<td>Reset</td>
</tr>
</tbody>
</table>

1. The addresses of these pseudo vectors are selected randomly for security reasons. See the following section on security against unauthorized access.
THE ICP PROCEDURE

Using the ICP scheme, assuming the HID is a keyboard, the following would be the procedure for reprogramming the JB8 user code:

1. With the keyboard plugged to a PC, the user initiates an ICP event by launching a program on the PC. This program clears the ICP_FLAG word to zero in the JB8.
2. User unplugs and replugs the USB connector.
3. After replugging, the JB8 detects that ICP_FLAG word is not a checksum and continues to run the ICP code. The PC detects the keyboard is in ICP mode, ready for firmware upgrade.
4. User launches a firmware upgrade program on the PC. (A separate keyboard must be used for this, since the keyboard in question is in ICP mode.)
5. To prevent unauthorized access, the PC program asks for the 8-byte security passcode.
6. Once pass security, the user is allowed to erase and program the user code in the JB8.
7. After user code upgrade, the final step is to program the ICP_FLAG word checksum.
8. User unplugs and replugs the USB connector.
9. After replugging, the JB8 detects that ICP_FLAG word is a checksum, and continues to run the user code — the normal operating mode.

USING THE ICP CODE

This section describes the ICP code listing in the APPENDIX: Code Listing.

After a reset, the value in the reset vector $FFFE:$FFFF points to $F800, the start of the ICP program. Once initialization has completed, the ICP code checks for conditions for entry into normal mode (the user code) or ICP mode (the ICP code).

JB8 will enter ICP mode when:

- The high byte of the pseudo reset vector ($FF7C) is invalid; i.e. it is not in range of the user FLASH area ($DC to $F7); or
- The ICP_FLAG word is not a checksum.

If neither of the two conditions is true, then JB8 enters normal operating mode.
Figure 3 shows the flow of the ICP code.

![Figure 3. ICP Program Flow](image)

Table 1 shows the mode entry conditions.

<table>
<thead>
<tr>
<th>Content of $FF7D</th>
<th>ICP_FLAG</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not $DC to $F7</td>
<td>Don’t care</td>
<td>ICP mode.</td>
</tr>
<tr>
<td>Don’t care</td>
<td>Not checksum</td>
<td>User mode</td>
</tr>
<tr>
<td>$DC to $F7</td>
<td>$01</td>
<td>User mode</td>
</tr>
</tbody>
</table>

When the JB8 is programmed only with the ICP code in place, the high byte of the pseudo reset vector at $F7FE equals $FF. This will cause the ICP code to continue to run in ICP mode. The user code can be programmed using the ICP functions.
After the user code is programmed, the high byte of the pseudo reset vector is in the valid range (between $DC and $F7) and the ICP_FLAG word is programmed with the checksum (checksum cannot be $0000). After an unplugged replug, the ICP code jumps to the user code for normal operation.

There are two ways for the JB8 to re-enter ICP mode:

- Program the ICP_FLAG word to $0000; or
- Pull PTA0 pin to logic 0.

The user code may include a specific command to program the ICP_FLAG. Once the ICP_FLAG is programmed with zero, the JB8 enters ICP mode when the device is re-plugged.

The ICP code supports limited USB standard requests as listed below:

- Get Descriptor
- Get Status
- Set Address
- Set Configuration
- Clear Feature

It has defined some vendor-specific requests as below:

**Table 3. Vendor-Specific Requests**

<table>
<thead>
<tr>
<th>Command</th>
<th>BmRequest Type</th>
<th>bRequest</th>
<th>wValue</th>
<th>wIndex</th>
<th>wLength</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Row</td>
<td>$40</td>
<td>$81</td>
<td>Start Address</td>
<td>End Address</td>
<td>Data Length</td>
<td>Data</td>
</tr>
<tr>
<td>Erase Block</td>
<td>$40</td>
<td>$82</td>
<td>Start Address</td>
<td>End Address</td>
<td>$00</td>
<td>$00</td>
</tr>
<tr>
<td>Verify Row</td>
<td>$40</td>
<td>$87</td>
<td>Start Address</td>
<td>End Address</td>
<td>Data Length</td>
<td>$00</td>
</tr>
<tr>
<td>Get Result</td>
<td>$C0</td>
<td>$8F</td>
<td>Start Address</td>
<td>End Address</td>
<td>$01</td>
<td>Result</td>
</tr>
</tbody>
</table>

The above vendor-specific requests provide the necessary commands to erase, program, and verify the user FLASH area.

One byte result will be returned duration the Get_Status command. The result indicates whether the last commands of Program_Row, Erase_Block or Verify_Row is successful.

- Success if result is $01
- Failure if result is $04
Programming the ICP_FLAG

Since the JB8 is designed for HID applications, it is better to use the HID command to program the ICP_FLAG (Set_ICP_Flag) so that no extra driver is needed. One example is to use the HID Set_Feature report with 8 bytes of data as shown in Table 4 to perform this function. The result is acknowledged by using the HID Get_Feature report of 8 bytes of data (Get_Ack), but only one byte is used.

Table 4. Feature Report Data

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1</td>
<td>Data 2</td>
<td>Data 3</td>
<td>Data 4</td>
<td>Data 5</td>
<td>Data 6</td>
<td>Data 7</td>
<td>Data 8</td>
</tr>
</tbody>
</table>

The 8 bytes of data (Data 1 to Data 8) used in Set_ICP_Flag is for security reasons. The command is valid only if the 8 bytes of data match the specific 8 bytes of stored in the JB8. One example is the 8 bytes of data at JB8’s $FFE6 to $FFED. After receiving the Set_ICP_Flag command with valid data the ICP_FLAG will be programmed to zero.

The acknowledgment is returned through data 1 of the Get_Feature report. Where:

- Success if acknowledgment is $00
- Fail if acknowledgment is $01

Command Example

Set_ICP_Flag:

<table>
<thead>
<tr>
<th>Commands</th>
<th>Data</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Report (Feature)</td>
<td>SETUP [21, 09, 00, 03, 01, 00, 08, 00]</td>
<td>Host sends out Set Report (Feature)</td>
</tr>
<tr>
<td></td>
<td>DATA0 [XX, XX, XX, XX, XX, XX, XX]</td>
<td>Host sends out 8 bytes of specific data</td>
</tr>
</tbody>
</table>

Get_Ack:

<table>
<thead>
<tr>
<th>Commands</th>
<th>Data</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Report (Feature)</td>
<td>SETUP [A1, 09, 00, 03, 02, 00, 08, 00]</td>
<td>Host sends out Get Report (Feature)</td>
</tr>
<tr>
<td></td>
<td>DATA0 [00, XX, XX, XX, XX, XX, XX]</td>
<td>Host sends out 8 bytes of specific data with data1 = $00</td>
</tr>
</tbody>
</table>

---

For More Information On This Product, Go to: www.freescale.com
Putting data $00$ to $3F$ to the FLASH location $DE00$ to $DE3F$:

<table>
<thead>
<tr>
<th>Commands</th>
<th>Data</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erase Block</td>
<td>SETUP [40, 82, 00, DE, FF, DF, 40, 00]</td>
<td>Erase a Block of $DE00$ - $DFFF$</td>
</tr>
<tr>
<td>Get Result</td>
<td>SETUP [C0, 8F, 00, 00, 00, 01, 00] DATA0 [01]</td>
<td>Host sends out Get_Restult Device returns result success</td>
</tr>
<tr>
<td>Program Row</td>
<td>SETUP [40, 81, 00, DE, 3F, DE, 40, 00] DATA0 [00, 01, 02, 03, 04, 05, 06, 07] DATA1 [08, 09, 0A, 0B, 0C, 0D, 0E, 0F] DATA1 [38, 39, 3A, 3B, 3C, 3D, 3E, 3F]</td>
<td>Host sends out Program_Row Host sends out 64 byte data of $00$ to $3F$</td>
</tr>
<tr>
<td>Get Result</td>
<td>SETUP [C0, 8F, 00, 00, 00, 01, 00] DATA0 [01]</td>
<td>Host sends out Get_Restult Device returns result success</td>
</tr>
</tbody>
</table>

**DEMO 1: Installing The USB ICP Driver**

The USBICP.EXE program requests the USBICP.SYS driver. Below shows the procedure for installation.

1. Plug in device with ICP program inside.
2. Click Next when the Add New Hardware Wizard window appears.
3. Select *Search for the best driver for your device* and then click *Next.*

![Add New Hardware Wizard](image1)

4. Specify the directory containing the `USBICP.INF` file and then click *Next.*

![Add New Hardware Wizard](image2)
5. Use the driver for JB8 Freescale IP Device and then click Next.

![Add New Hardware Wizard](image1)

6. Click Next.

7. Locate the directory containing the `USBICP.SYS` driver if you are told to do so.

8. Finished.

---

**DEMO 2: Running USBICP**

1. Open `USBICP.EXE` and select the parametric file `JB8ICP_END.IMP`.

![Open File](image2)
USBICP program window appears.

2. Erase FLASH and then do Blank Check (skip for first time programming, i.e. FLASH user area is blank).

3. Select the file to be programmed (e.g.: JB8-USB.SX)

4. Select Program device and then select Verify.
DEMO 3: Running SETICP.EXE

1. Run FreescaleHID.exe.

![FreescaleHID Demo](image)

2. Select SetICP (kbd, mse) (change Vendor ID and Product ID if necessary).

![Set ISP with Security Codes](image)

3. Change ICP security code if necessary and then click OK.
4. Unplug and replug to cause the device to enter ICP mode.
FURTHER INFORMATION

MC68HC908JB8 Technical Data,
Freescale document number: MC68HC908JB8/D.
APPENDIX: Code Listing

*******************************************************************************
*******************************************************************************
* Copyright (c) 2002
* File Name: JB8_ICP.ASM
*
* Purpose: JB8_ICP is a pre-loaded firmware that allows user to do
* the firmware upgrade through the USB interface
*
* Assembler: CodeWarrior
* Version: 2.1
*
* Description: See below.
*
* Author: Location: First release date:
*
* Current Release Level: 1st released version
*
* Last Edit Date: 2002.10.10
*
* UPDATE HISTORY:
* Rev YY/MM/DD Author Description of Change
* ----- ---------- ------------ -----------------------
* 0.1 00/03/17 Bruce Ding LD64 2nd silicon Monitor Code
* Keny Chen
* 0.2 01/05/02 Bruce Ding Changed for 908JB16
* Alu Lin Removed check valid address;
* Derek Lau Removed Read_block routine;
* Added Option to disable USB_ICP
* and serial monitor mode.
* 1.0 02/10/10 Derek Lau Modified for JB8.
*******************************************************************************
*******************************************************************************

Freescale reserves the right to make changes without further notice to any
product herein to improve reliability, function, or design. Freescale
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.
Freescale 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 Freescale product could create a situation
where personal injury or death may occur. Should Buyer purchase or use
Freescale products for any such intended or unauthorized application,
Buyer shall indemnify and hold Freescale 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 was negligent regarding the design or
manufacture of the part. Freescale and the Freescale logo are registered trademarks of Freescale Semiconductor, Inc.
*******************************************************************************
;* Parameter Equates
;
  include "jb8-eqs.h" ; jb8 registers definitions
  include "macro8-asm.h" ; 08 CPU macro

ICP_BUF_SIZE equ $40 ; maximum buffer size

;*************************************
;* Variables Definition
;*************************************

ORG RAM_BEG+8

ICP_RAM_BEG:

V_ChkSumH equ * ; checksum high byte
V_CtrByte ds 1 ; control byte for erase
b_MASSBIT equ 6 ; mass erase bit in FLCR
V_CPUSpeed ds 1 ; CPU speed = CPU bus x 4
V_LAddr ds 2 ; last address for programming
Q_RAM_Blk_Erase equ * ; block erase program in RAM
Q_Work_Buf equ * ; data buffer for Control Pipe
Q_ICP_Buf ds ICP_BUF_SIZE ; 64 byte buffer

UICP_RAM_BEGIN:

Q_Setup_Buf equ *

VI_bmReqType ds 1 ; Characteristic of Request
b_Rcpt0 equ 0 ; Recipient=$00 Device
b_Rcpt1 equ 1 ; =$01 Interface
b_Rcpt2 equ 2 ; =$02 Endpoint
b_Rcpt3 equ 3 ; =$03 Other
b_Rcpt4 equ 4 ; =$04-31 <reserved>

b_Type0 equ 5 ; Req. Type=0 Standard
b_Type1 equ 6 ; =1 Class
b_Type2 equ 7 ; =2 Vendor
b_Type3 equ 8 ; =3 <reserved>

VI_bRequest ds 1 ; Request Code

V_wValue_L ds 1 ; Value Field for the request
V_wValue_H ds 1

V_wIndex_L ds 1 ; Index Field for the request
V_wIndex_H ds 1

V_wLength_L ds 1 ; no. of bytes in Data Stage
V_wLength_H ds 1

V_Transaction ds 1 ;0:IDLE 1:SETUP 2:OUT 3:IN

; content definition
TRF_IDLE equ 0
TRF_SETUP equ 1
TRF_OUT equ 2
TRF_IN equ 3

;* ---------------


V_UDR_Size ds 1
V_Config_Value ds 1

;*--------------------------------------------------------------------
V_TRF_Status ds 1
b_ADDR_SET equ 0
b_WAIT_ADDR equ 1
b_ST_WAIT equ 2
b_ST_TYPE equ 3
b_OUT_DONE equ 4

; ADDR_BIT equ $01 ; 1:device address is assigned
ADDR_WAIT equ $02 ; 1:ADDR_request is waiting for status stage
STATUS_WAIT equ $04 ; 1:status_stage is waiting
STATUS_TYPE equ $08 ; 1:IN status stage
DO_USBOUT equ $10 ; 1:OUT data stage is done::usb_proc()

;*--------------------------------------------------------------------

;usb_status
V_Rx_Cnt ds 1
V_Tx_Cnt ds 1
V_Rx_Ptr ds 1
V_Tx_Ptr ds 1
V_UDR_Ptr ds 1

;*--------------------------------------------------------------------
V_Toggle_Buf ds 1
SEQ_MASK equ $80
b_SEQ_BIT equ 7

;* ====== Variables for ICP ===========================================
V_ICP_CMMD ds 1
b_PROG_Set equ 0
b_Erase_Set equ 1
b_Mass_Erase equ 2
b_Read_Set equ 3
b_Verify_Set equ 4
b_Do_Read equ 6
b_Data_OK equ 7

;*--------------------------------------------------------------------

;* --- [RAM Routine] copy content of VD_Opd1[x] to VD_Opd2 ---
;* -- called in GET_DESC() --
; equivalent to : lda <VD_Opd1_H:L>,X
; sta <VD_Opd2 H:L>,X
;
; D_LONG_LDAX equ * ; <for Device Command Handler>
VI_LDA ds 1 ; Opcode of LDA(16-bit Idx) = $D6
VI_Opd1_H ds 1 ; Offset(High byte)
VI_Opd1_L ds 1 ; Offset(Low byte)
VI_STA ds 1 ; Opcode of STA(direct) = $D7
VI_Opd2_H ds 1 ; Offset(High)
VI_Opd2_L ds 1 ; Offset(Low)
VI_RTS ds 1 ; Opcode of RTS = $81
Var_End equ *
;* parameters to pass into ICP subroutine
;*
START_ADD equ *
V_Start_Add_H ds 1 ; MSB FLASH start address
V_Start_Add_L ds 1 ; LSB FLASH start address
;
END_ADD equ *
V_End_Add_H ds 1 ; MSB FLASH ending address
V_End_Add_L ds 1 ; MSB FLASH ending address
;
MONITOR_VERIFY equ $FC03 ; Monitor routine for verify
MONITOR_PROGRAM equ $FC09 ; Monitor routine for programming
;
V_Source ds 1
V_Destination ds 1
;
UICP_RAM_END equ *
;
UICP_RAM_SIZE equ UICP_RAM_END-UICP_RAM_BEGIN
;
;*****************************************************************************
;* CONSTANT DEFINITION
;*
NUM_BLK equ !16 ; Number of USB block for a Flash block
FEATURE_SIZE equ 8 ; block size of programming command
;
;* ===============================
;* Device/Endpoint Feature Select
;* ===============================
EP_STALL equ 0
RM_WAKEUP equ 1
;
;* ===============================
;* Descriptor types
;* ===============================
DEVICE_TYPE equ 1
CONFIG_TYPE equ 2
STRING_TYPE equ 3
INTERFACE_TYPE equ 4
ENDP_TYPE equ 5
HID_TYPE equ $21
REPORT_TYPE equ $22
;
;* ===============================
;* HID ReportType
;* ===============================
HID_INPUT equ 1
HID_OUTPUT equ 2
HID_FEATURE equ 3
;
INPUT_TYPE equ 1
OUTPUT_TYPE equ 2
FEATURE_TYPE equ 3
; * =========================================================
;     bRequest
; * =========================================================
;
; * ----------------------
; * Standard Request
; * ----------------------
GET_STATUS       equ  0
CLR_FEATURE      equ  1
SET_FEATURE      equ  3
SET_ADDR         equ  5
GET_DESCRIPTOR   equ  6
SET_DESCRIPTOR   equ  7
GET_CONFIG       equ  8
SET_CONFIG       equ  9
GET_INTERFACE    equ  !10
SET_INTERFACE    equ  !11
SYNCH_FRAME      equ  !12
;
; * ----------------------
; * HID Class Request
; * ----------------------
GET_REPORT       equ  1
GET_IDLE         equ  2
SET_REPORT       equ  9
SET_IDLE         equ  $OA
;
; * ----------------------
; * USB ICP Request
; * ----------------------

; PROG BLOCK CMMD -
; { %01000000,$81,Start_Adr_L,Start_Adr_H,End_Adr_L,End_Adr_H,$40,$0 }
;
; ERASE BLOCK CMMD -
; { %01000000,$82,Start_Adr_L,Start_Adr_H,End_Adr_L,End_Adr_H,$40,$0 }
;
; ERASE ALL CMMD -
; { %01000000,$83,$0,$0,$0,$0,$0,$0 }
;
; READ BLOCK CMMD -
; { %01000000,$84,Start_Adr_L,Start_Adr_H,End_Adr_L,End_Adr_H,CMD_Length }
;
; GET_INFO CMMD -
; { %1000000,$85,$0,$0,$0,$0,$8,$0 }
;
; EXIT_ICP CMMD -
; { %1000000,$86,$0,$0,$0,$0,$0,$0 }
;
; VERIFY_CODE CMMD -
; { %1000000,$87,Start_Adr_L,Start_Adr_H,End_Adr_L,End_Adr_H,$40,$0 }
;
; GET_STATUS CMMD -
; { %1100000,$8F,$0,$0,$0,$0,$1,$0 }
In-Circuit Programming of FLASH Memory via the USB for the MC68HC908JB8

SBB_PROG equ $81
SBB_ERASE equ $82
ERASE_ALL equ $83
SBB_READ equ $84
VERIFY_CODE equ $87
GET_ICP_STATUS equ $8F

;****************************************************************************
;* Return: Acc = $AF if erase/program succeeds
;* Acc = $5F if erase/program fails
;****************************************************************************

DMCR equ $0016
ALIF equ $0007
NAKIF equ $0006
MAST equ $0004
DADR equ $0017
DEN equ $0007
DCR equ $0018
DSR equ $0019
RXIF equ $0007
TXIF equ $0006
MATCH equ $0005
SRW equ $0004
TXBE equ $0001
DDTR equ $001A
DDRR equ $001B
D2ADR equ $001C
PDCR equ $0069 ; to fix 1st version bug (000920 bruce+)

ICP_ADDRESS equ $0036
MCU_ADDRESS equ $0034
ACK_SIGNAL equ $00AF
NAK_SIGNAL equ $005F
NOACK_SIGNAL equ $005F

CODE_VER equ $005A
CODE_PROG equ $0055
CODE_ME equ $00A5
CODE_BE equ $00AA
CODE_EXIT equ $0099

USE_USB_IPULLUP set 0 ; 0 - use internal pullup

myCode SECTION Short

;****************************************************************************
;* Main Program
;****************************************************************************

For More Information On This Product,
Go to: www.freescale.com
ICP_Reset_Init:  
  _Startup: 
  lda JMP_Reset_Init+1 ; check app address valid  
  cheqa #$FF,USB_ICP ; usb ICP if app address blank  
  KCMPLO (ROM_BEG/256),USB_ICP ; usb ICP if app address invalid  
  clr V_ChksumH ; clear checksum high byte  
  clra ; clear ACC for cal checksum  
  ldhx #$F600 ; checksum start address  
  ChkSum_Loop: 
  add ,x ; add the bytes in flash  
  bcc Not_Overflow ; overflow ?  
  inc V_ChksumH ; increase checksum high byte if yes  
  Not_Overflow: 
  aix #1 ; increase flash address  
  cphx #(ICP_FLAG) ; flash address reaches ICP_FLAG  
  bne ChkSum_Loop ; continue if not finished  
  add ICP_FLAG+1 ; sum of flash + ICP_FLAG low byte  
  bcc Not_Overflow1 ; overflow ?  
  inc V_Chksum ; increase checksum high byte if yes  
  Not_Overflow1: 
  tsta ; checksum low byte+ICP_FLAG low byte=0 ?  
  bne USB_ICP ; ICP mode if sum <> 0  
  lda ICP_FLAG ; get ICP_FLAG high byte  
  add V_Chksum ; add checksum high byte  
  bne USB_ICP ; ICP mode if sum <> 0  
  ;
  Jmp_Application  
  jmp JMP_Reset_Init ; jmp to application program  
  ;
  ;***********************************************************************  
  ;* USB_ICP  
  ;***********************************************************************  
  ;====================================================================  
  ; USB Initialization  
  ;====================================================================  
  USB_ICP: 
  ldhx #(RAM_END+1) ; set SP end of RAM  
  txs  
  mov #$00000001,CONFIG ; disable COP, enable STOP  
  sei  
  clrh ; reset high byte of H:X  
  ;
  ;* Initialize the USB module  
  ;
  ;* USB Initialization  
  ;==============================================================================================  
  ITS_USB_ICP: 
  bsr RST_USB_SIE ; init and enable USB module  
  mov #!12,V_CPUSpeed ; V_CPUSpeed = 4 * 3  
  lda #$F8 ; unprotect FLASH  
  sta FLBPR
;* Clear Page Zero RAM area
;*
;* Set up RAM routine
;* 

ldx #UICP_RAM_SIZE
CLR_RAM_L:
clr (UICP_RAM_BEGIN+1),x
dbnzx CLR_RAM_L

mov #$D6,VI_LDA
mov #$D7,VI_STA
mov #$81,VI_RTS

;====================================================================
; Main Loop
;====================================================================
MAIN_LOOP_ICP:

;*-------------------------------------------------------------------
brclr b_OUT_DONE,V_TRF_Status,END_PROC_OUT
bclr b_OUT_DONE,V_TRF_Status

brset b_PROG_Set,V_ICP_CMMD,GOT_PROG_BLK
bsr CODE_VERIFY ; Verify a Flash Block
bra END_DATA_OK

GOT_PROG_BLK:
jsr PROG_CODE ; Program a Flash Block

END_DATA_OK:
jsr CHECK_RESULT

END_PROC_OUT:

;*-------------------------------------------------------------------
TEST_RX:
brclr b_RXD0F,UIR1,test_tx ;[H/W error-free Setup/OUT transaction]
bset b_RXD0FR,UIR2 ; Clear RXD flag

;* It's an OUT token
;*-------------------------------------------------------------------
bset b_TSTOP,TSC ; timer stop (no more timeout for usb)
jsr RX_INT

;*-------------------------------------------------------------------
test_tx:
brclr b_TXD0F,UIR1,TEST_NULL ;[H/W error-free IN transaction]
bset b_TXD0FR,UIR2 ; Clear TXD flag
In-Circuit Programming of FLASH Memory via the USB for the MC68HC908JB8

Freescale Semiconductor, Inc.

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

In-Circuit Programming of FLASH Memory via the USB for the MC68HC908JB8

`;* ====================
`;* It's an IN token
`;* ====================
jsr TX_INT
;
`;* ====================
`;* Nothing happened
`;* ====================
TEST_NULL:
   bra MAIN_LOOP_ICP ; loop while timer not overflow
;
`;* RST_USB_IF - initialize USB module
;* RST_USB_SIE:
;ifeq USE_USB_IPULLUP
mov #%00000100,UCR3 ; enable internal D- pullup
endif
mov #$80,UADDR ; restore default addr($00), enable USB
clr UIR0
mov #$0010000,UCR0 ; enable EP0 rx
clr UCR1
clr UCR2
mov #$10111111,UIR2 ; clear int. flags
rts
;
`;* Program + Verify
; Input: Flash address = START_ADD (2 bytes), END_ADD (2 bytes)
; Data Buffer address  = $004C - $008B (max 64 bytes)
; Usage:
; Output: Acc = #ACK_SIGNAL (ok)
; Acc = #NOACK_SIGNAL (fail)
;`prog_code
;
*========================================================
* Variables for Flash Program routines
*========================================================
ldhx END_ADD
sthx V_LAddr
ldhx START_ADD
jsr MONITOR_PROGRAM ; Program FLASH in monitor code
;======================================================================
; Verify
; Input: Flash address = START_ADD (2 bytes), END_ADD (2 bytes)
; Data address = $0100 - $02FF (max 512 bytes)
; Usage: START_ADD (2 bytes), SOURCE_INX (2 bytes), TARGET_ADD (2 bytes)
; Output: Acc = #ACK_SIGNAL   (ok)
;         Acc = #NOACK_SIGNAL (fail)
;======================================================================

CODE_VERIFY:

ldhx END_ADD
lda END_ADD
KCMPHI $F7,PROG_FAIL ; fail if invalid address
sthx V_LAddr
ldhx START_ADD
jsr MONITOR_VERIFY
bcc PROG_FAIL

PROG_OK:

lda #ACK_SIGNAL ; y --> ACK to host
rts ; return

PROG_FAIL:

lda #NOACK_SIGNAL ; n --> fail
rts

;======================================================================
; Block Erase
; Input: Flash address = START_ADD (2 bytes)
; Usage: SOURCE_INX (2 bytes)
; Output: Acc = #ACK_SIGNAL   (ok)
;         Acc = #NOACK_SIGNAL (fail)
;======================================================================

BERASE:

bsr BlkErase2RAM ; copy block erase routine to RAM
ldhx START_ADD
lda #(1<<b_ERASE) ; MUST load Acc with b_ERASE
jsr Q_RAM_Blk_Erase ; execute block erase in RAM
lda #ACK_SIGNAL ; ok
rts

*-------------------------------------------------------------
BlkErase2RAM:

ldx #Blk_Erase.Len ; get blk erase routine length

BE2RAM1:

lda Block_Erase-1,x ; load from FLASH
sta Q_RAM_Blk_Erase-1,x ; copy to RAM
rts
In-Circuit Programming of FLASH Memory via the USB for the MC68HC908JB8

For More Information On This Product,
Go to: www.freescale.com
NOTES:
How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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

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

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

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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document. Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters 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.