Application Note

AN2399/D
Rev. 0, 3/2003

In-Circuit Programming of FLASH Memory via the Universal Serial Bus for the MC68HC908JB16

By Derek Lau
Applications Engineering
Microcontroller Division
Hong Kong

This application note describes a method of in-circuit programming of FLASH memory via the Universal Serial Bus for the MC68HC908JB16.

For detailed specification on MC68HC908JB16 device, please refer to the data sheet; Motorola order number: MC68HC908JB16/D.

INTRODUCTION

The Motorola MC68HC908JB16 (hereafter referred as JB16) is a member of the HC08 Family of microcontrollers (MCUs). The features of the JB16 include a Universal Serial Bus (USB) interface and dual PLL clock generators, which make this MCU suited for 27MHz wireless personal computer Human Interface Devices (HID), such as wireless mouse and keyboard receivers.

On the JB16, 16k-bytes of FLASH memory is allocated for the user code, with an additional 32-bytes for user defined reset and interrupt vectors. A high voltage supply is not required by the JB16 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.

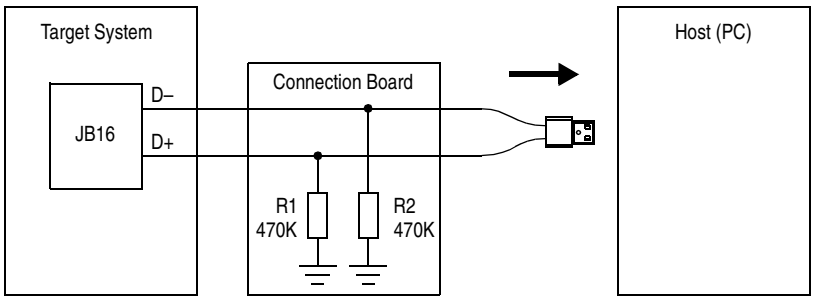
This application note describes a method of implementing ICP using the USB as the communication link between host (PC) and HID keyboard.

The JB16 has an advantage over previous HC08 devices with a USB module, such as the JB8 (MC68HC908JB8). Routines embedded in the monitor ROM area ($FA00–$FDFF and $FE10–$FFCF) are available to simplify the ICP process. A USB communications handler is already in the monitor ROM.

PROGRAMMING A BLANK JB16

The usual method of programming the blank (erased state) FLASH memory in a MCU is to use a dedicated programmer. This usually involves placing the device into a socket for the programming operation. After programming, the device is then soldered onto the target system.

For the JB16, ICP for a blank FLASH is possible. A blank JB16 can be soldered onto the target system prior to any programming. To use the USB for ICP, follow the connections shown in **Figure 1**.



NOTES:
R1 and R2 can be placed on the target system instead of the connection board.
R2 is optional if an external 1.5K pullup is used on D–.

Figure 1. USB ICP Mode for a Blank JB16

With the resistors R1 and R2 connected, a blank JB16 enters USB ICP mode after a power-on reset. With the appropriate drivers installed, the host will recognize an ICP request when the target system is plugged to the host. The USB commands used are described in later sections of this application note.

For ease of re-programmability after initial programming, the user code for the JB16 must contain a mechanism to re-enter USB ICP mode for user code upgrade/changes while in normal use. The following sections will describe how to modify the user code for USB ICP for non-blank JB16's.

PROGRAMMING A NON-BLANK JB16

To use the USB interface as a communications link for ICP in user mode, the user code in the JB16 must include routines to initiate the ICP process.

The following method uses code (the ICP determination code) that is programmed in the last block of JB16 FLASH memory to determine whether the target system enters user mode or ICP mode.

Figure 2 shows the FLASH memory usage for the JB16 ICP scheme.

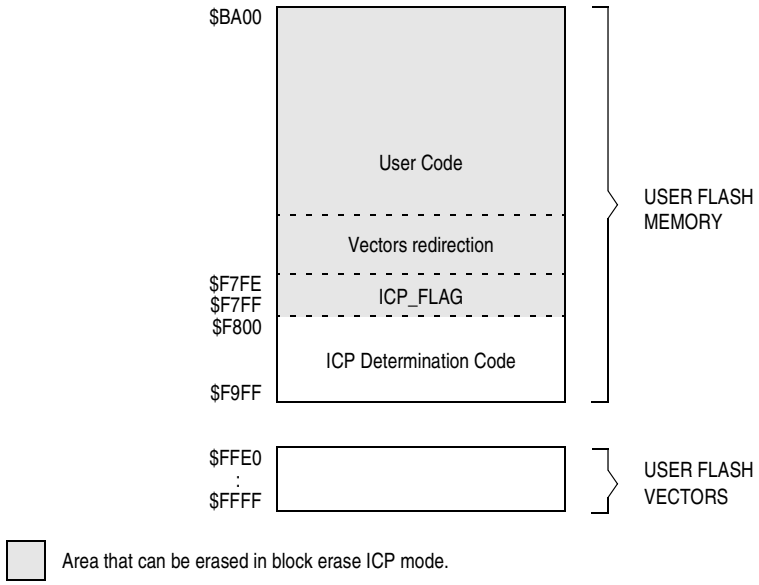


Figure 2. FLASH Memory Usage for ICP

From **Figure 2**, the user block ranges from $BA00 to $F9FF, and the user vectors block ranges from $FFE0 to $FFFF.

Both *block erase* mode and *mass erase* mode are implemented in this ICP scheme. In ICP block erase mode, the ICP determination code ($F800–$F9FF) and the user FLASH vectors ($FFE0–$FFFF) do not get reprogrammed. Only the FLASH from $BA00 to $F7FF (the shaded area shown in **Figure 2**) gets erased and programmed. In ICP mass erase mode, the entire FLASH gets erased and programmed.

If block erase mode is used to modify the user code, the vectors at $FFE0–$FFFF may be affected, hence they must be redirected.

Vector Redirecting

For the ICP scheme to erase and reprogram the user code only (leaving the ICP determination code untouched), 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 protection mechanism allows only a mass erase operation to erase the user FLASH vectors).

Since the user FLASH vectors are now fixed, these must be redirected 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 3** shows how the vectors are redirected. The only vector that is not redirected is the reset vector. The reset vector always points to $F800 — the start of the ICP determination code.

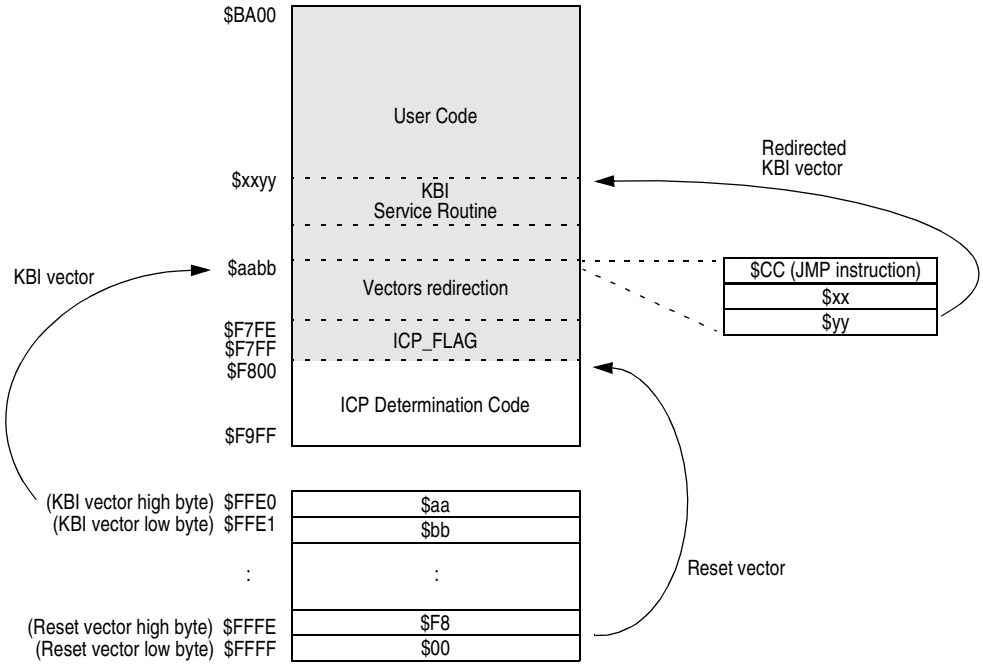


Figure 3. Vector Redirecting

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

Table 1. Vector Addresses

Vector Address	Pseudo Vector Address	Interrupt
$FFE0 : $FFE1	$F7DB : $F7DC	Keyboard
$FFE2 : $FFE3	$F7DE : $F7DF	SCI Transmit
$FFE4 : $FFE5	$F7E1 : $F7E2	SCI Receive
$FFE6 : $FFE7	$F7E4 : $F7E5	SCI Error
$FFE8 : $FFE9	$F7E7 : $F7E8	TIM2 Overflow
$FFEA : $FFEB	$F7EA : $F7EB	TIM2 Channel 01
$FFEC : $FFED	$F7ED : $F7EE	TIM2 Channel 1
$FFED : $FFEF	$F7F0 : $F7F1	TIM2 Channel 0
$FFF0 : $FFF1	$F7F3 : $F7F4	TIM1 Overflow
$FFF2 : $FFF3	$F7F6 : $F7F7	TIM1 Channel 01
$FFF4 : $FFF5	$F7F9 : $F7FA	TIM1 Channel 1
$FFF6 : $FFF7	Aw : Aw+1 (1)	TIM1 Channel 0
$FFF8 : $FFF9	Ax : Ax+1 (1)	IRQ
$FFFA : $FFFB	Ay : Ay+1 (1)	USB
$FFFC : $FFFD	Az : Az+1 (1)	SWI
$FFFE : $FFFF	$F7FC : $F7FD	Reset

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

Security Against Unauthorized Access

The contents of the 8 bytes, $FFF6 to $FFFD, are used as a passcode for entry into JB16'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 sixteen pseudo vectors were fixed locations, say in an array from $F7CF 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 is used for this purpose. The ICP_FLAG is a checksum that is the 1's compliment of the sum of the contents of FLASH memory from $F600 to $F7FD.

The ICP_FLAG

After reset, the ICP_FLAG word is read to determine whether the JB16 should enter normal operating mode or ICP mode. This word is at $F7FE:$F7FF; the last two bytes in the user code area. This use of the ICP_FLAG is explained in the subsequent sections.

THE ICP Procedure

Using the ICP scheme, assuming the HID is a keyboard, the following would be the procedure for reprogramming the JB16 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 JB16.
2. User unplugs and re-plugs the USB connector.
3. After re-plugging, the JB16 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 being upgraded 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 JB16.
7. After user code upgrade, the final step is to program the ICP_FLAG word checksum.
8. User unplugs and re-plugs the USB connector.
9. After re-plugging, the JB16 detects that ICP_FLAG word is a checksum, and continues to run the user code — the normal operating mode.

USING THE ICP DETERMINATION CODE

This section describes the ICP determination 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 determination code. Once initialization has completed, the ICP code checks for conditions for entry into normal mode (the user code) or ICP mode.

JB16 will enter ICP mode when:

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

If neither of the two conditions is true, then JB16 enters normal operating mode.

Figure 4 shows the flow of the ICP code.

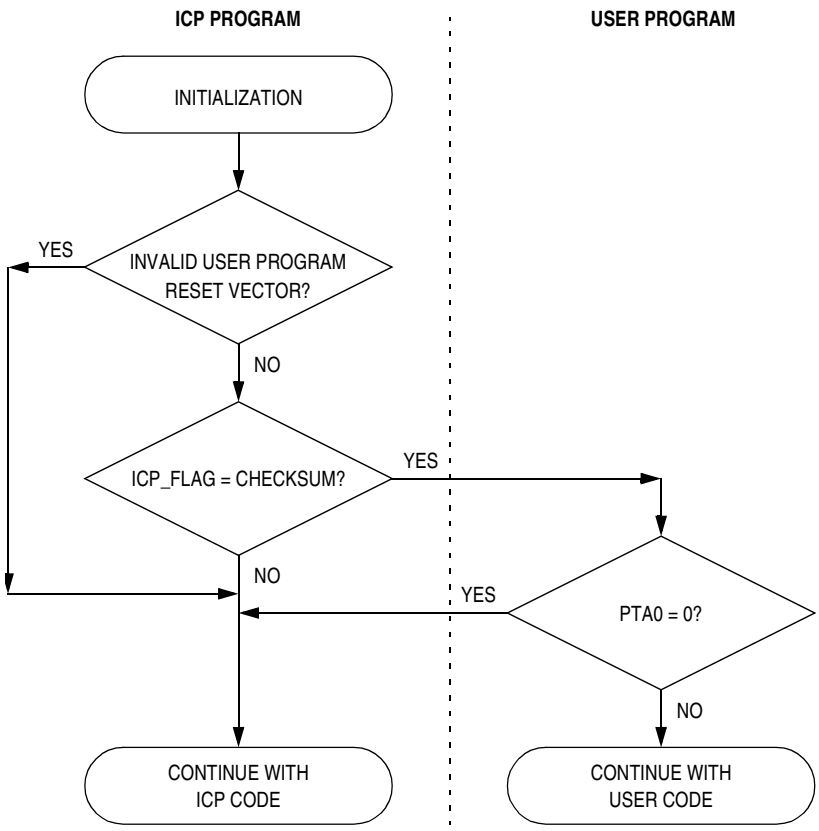


Figure 4. ICP Program Flow

Table 2 shows the mode entry conditions.

Table 2. Entry Conditions

Content of $FF7D	ICP_FLAG	Mode
Not $BA to $F7	Don't care	ICP mode.
Don't care	Not checksum	
$BA to $F7	Checksum	User mode

After the user code is programmed, the high byte of the pseudo reset vector is in the valid range (between $BA and $F7) and the ICP_FLAG word is programmed with the checksum (checksum cannot be $0000). After an unplug and re-plug, the ICP code jumps to the user code for normal operation.

There are two ways for the JB16 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 JB16 enters ICP mode when the device is re-plugged.

USB Commands

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

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

Table 3 shows some defined vendor-specific requests.

Table 3. Vendor-Specific Requests

Command	BmRequest Type	bRequest	wValue	wIndex	wLength	Data
Program Row	$40	$81	Start Address	End Address	Data Length	Data
Erase Block	$40	$82	Start Address	End Address	$0000	—
Mass Erase	$40	$83	$0000	$0000	$0000	—
Verify Row	$40	$87	Start Address	End Address	Data Length	Data
Get Result	$C0	$8F	Start Address	End Address	$01	Result

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 JB16 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

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8

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 JB16. One example is the 8 bytes of data at JB16's $FFD6 to $FFDD. 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 Examples

Set_ICP_Flag:

Commands	Data	Comment
Set Report (Feature)	SETUP [21, 09, 00, 03, 01, 00, 08, 00] DATA0 [XX, XX, XX, XX, XX, XX, XX, XX]	Host sends out Set Report (Feature) Host sends out 8 bytes of specific data

Get_Ack:

Commands	Data	Comment
Get Report (Feature)	SETUP [A1, 09, 00, 03, 02, 00, 08, 00] DATA0 [00, XX, XX, XX, XX, XX, XX, XX]	Host sends out Get Report (Feature) Host sends out 8 bytes of specific data with data1 = $00

Programing data $00, $01, $02, ...$3F to the FLASH location $DE00 to $DE3F:

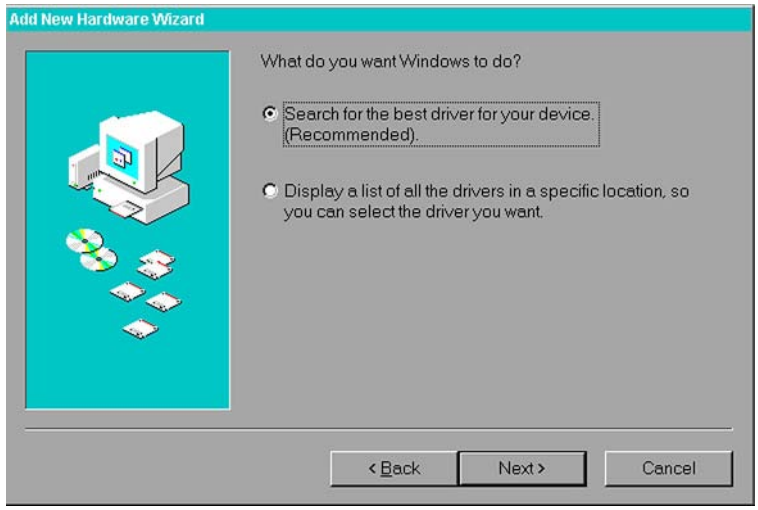
Commands	Data	Comment
Erase Block	SETUP [40, 82, 00, DE, FF, DF, 40, 00]	Erase a Block of $DE00–$DFFF
Get Result	SETUP [C0, 8F, 00, 00, 00, 00, 01, 00] DATA0 [01]	Host sends out Get_Result Device returns result success
Program Row	SETUP [40, 81, 00, DE, 3F, DE, 40, 00] DATA0 [00, 01, 02, 03, 04, 05,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

DEMO 1: Installing The USB ICP Driver

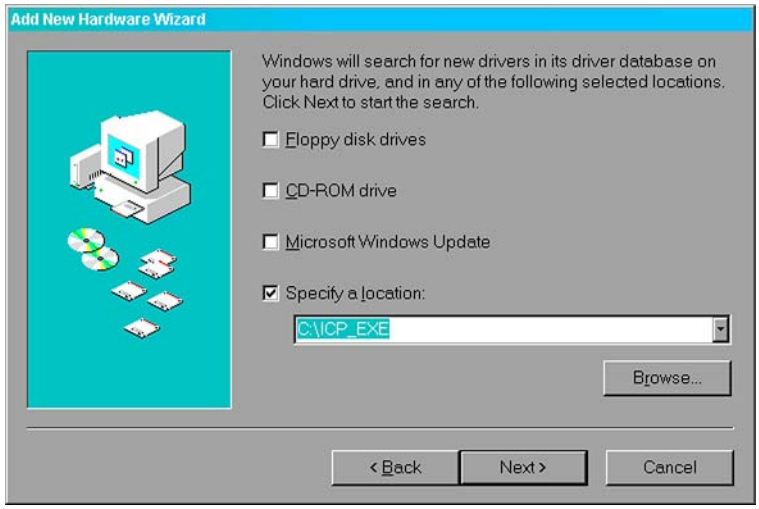
The USBICP.EXE program requests the USBICP.SYS driver. Below shows the procedure for installation.

1. Plug in device with ICP program inside.
2. Click *Next* when the Add New Hardware Wizard window appears.

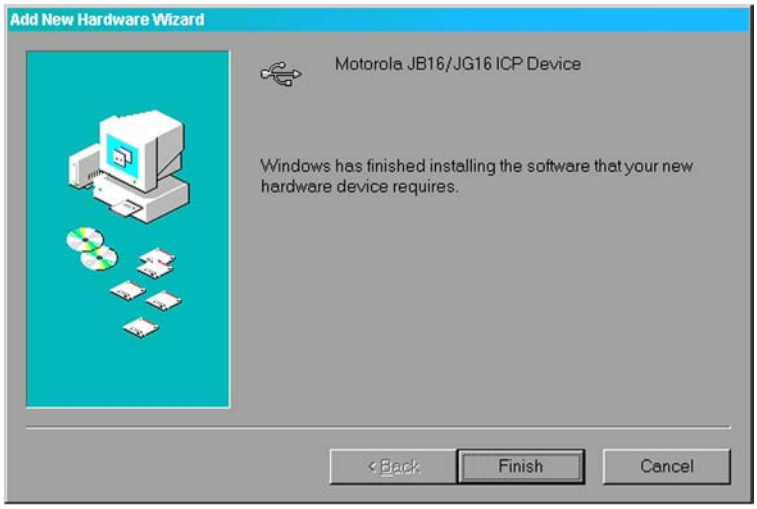
3. Select *Search for the best driver for your device* and then click *Next*.



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



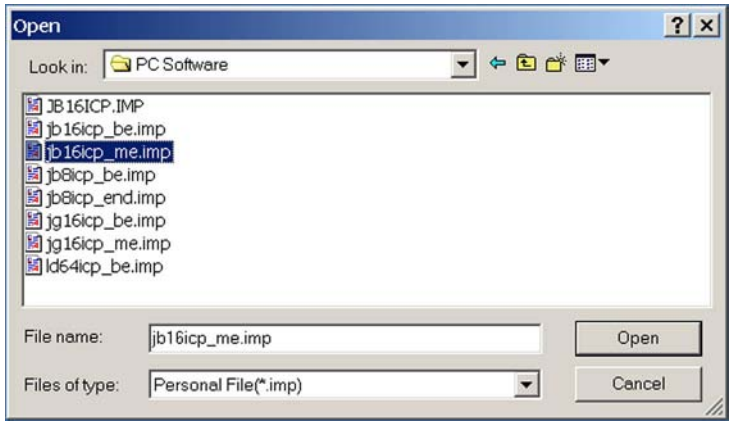
5. Use the driver for Motorola JB16 ICP Device and then click *Next*.



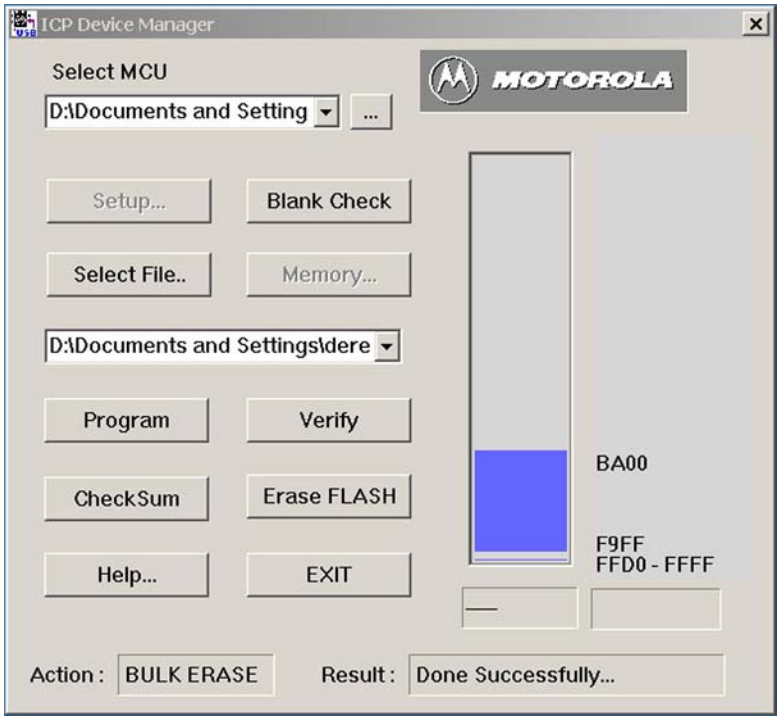
6. Click *Next*.
7. Locate the directory containing the `USBICP.SYS` driver if you are told to do so.
8. Finished.

DEMO 2: Running USBICP

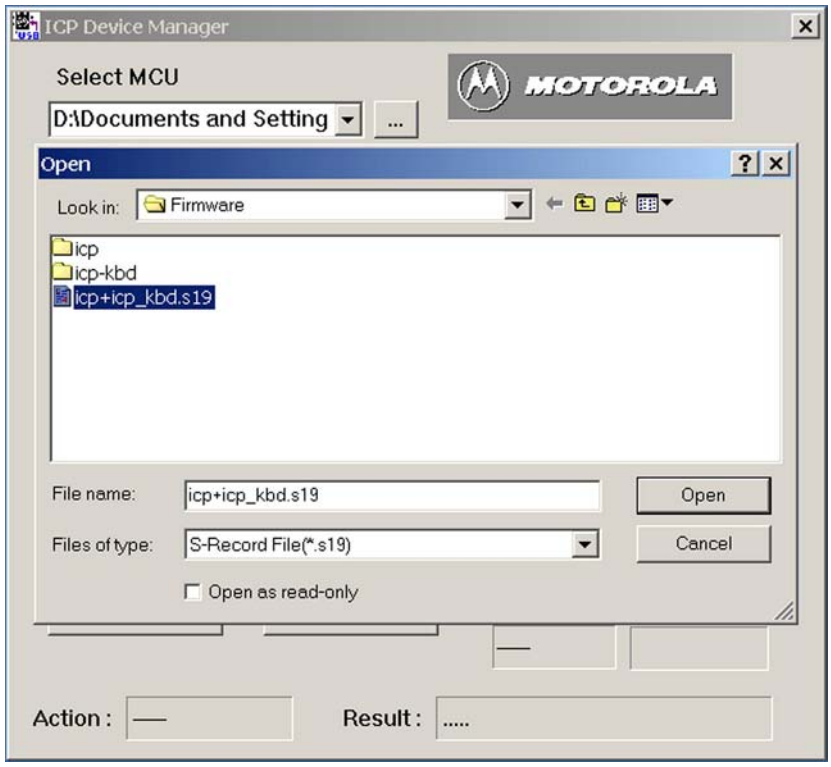
1. Open `USBICP.EXE` and select a parametric file:
   `jb16icp_me.imp` for mass erase scheme and program all area.
   `jb16icp_be.imp` for block erase scheme and program block.



`USBICP` program window appears.



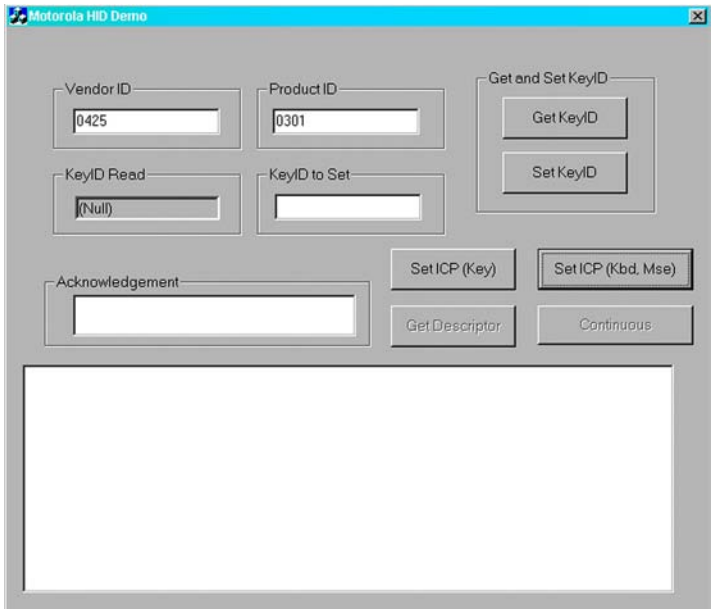
2. Erase FLASH and then do Blank Check (skip for first time programming, i.e. FLASH user area is blank).
3. Select the file to be programed (e.g.: `icp+icp_kbd.s19`)



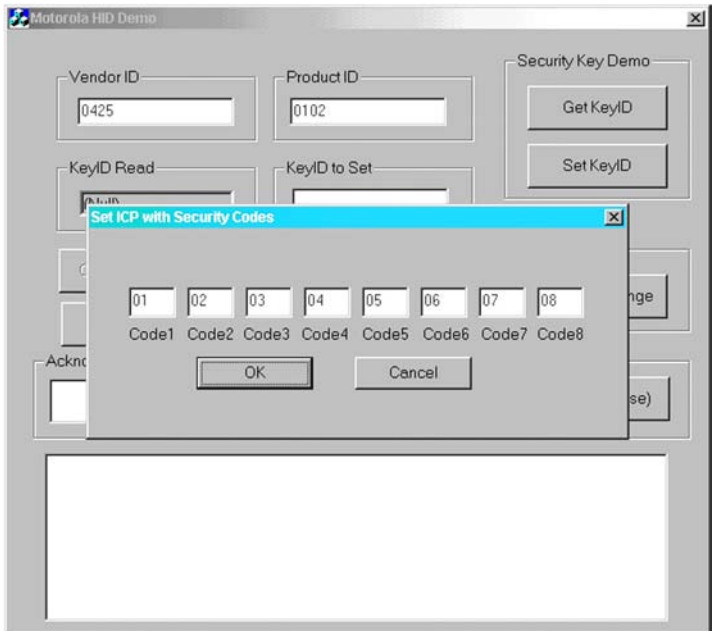
4. Select Program device and then select Verify.

DEMO 3: Running SETICP.EXE

1. Run `MotorolaHID.exe`.



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



3. Change ICP security code if necessary and then click OK.
4. Unplug and re-plug to cause the device to enter ICP mode.

FURTHER INFORMATION

MC68HC908JB16 Technical Data,
Motorola document number: MC68HC908JB16/D.

In-Circuit Programming of FLASH Memory via the USB for the MC68HC908JB8,
Motorola document number: AN2398/D.

APPENDIX: Code Listing

```
;*******************************************************************************
;* Copyright (c) Motorola 2002
;* File Name: JB16_ICP.ASM
;*
;* Purpose: JB16_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: Derek Lau     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
;*  -----  ----------   ------------     ----------------------
;*   1.0    02/01/02     Derek Lau       Original version
;*
;*******************************************************************************
;* 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, Inc.
;*******************************************************************************
;
;
;
;
;
;
```

```
;* Parameter Equates
;
        include "jb16-eqs.h"                    ; jb16 registers definitions
        include "macro8-asm.h"                  ; 08 CPU macro
;
MON_USB_ICP     equ     $FA19                   ; Monitor USB ICP routines
;
USE_USB_IPULLUP set     0                       ; 0 - use internal pullup
;
DEFAULT_RAM     SECTION SHORT
V_ChkSumH       ds.b    1
;
                XDEF    _Startup
myCode          SECTION Short
;
;*=========================================================
;*
;*      Main Program
;*
;*=========================================================
ICP_Reset_Init:
_Startup:
                lda     JMP_Reset_Init+1        ; check if app address valid
                cbeqa   #$FF,USB_ICP            ; usb ICP if app address blank
                KCMPLO  (ROM_BEG/256),USB_ICP   ; usb ICP if app address invalid

;               jmp     JMP_Reset_Init          ; for testing only
;
                clr     V_ChkSumH               ; clear checksum high byte
                clra                            ; clear acc for calculate checksum
                ldhx    #$F600                  ; checksum starting 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 finish
;
                add     ICP_FLAG+1              ; sum of flash + ICP_FLAG low byte
                bcc     Not_Overflow1           ; overflow ?
                inc     V_ChkSumH               ; 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_ChkSumH               ; 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_ICP:
                sei
                mov       #%00000011,CONFIG
;
                IFEQ      USE_USB_IPULLUP
                mov       #%00000100,UCR3          ; enable USB pullup
;
ENDIF
                jmp       Mon_USB_ICP
;
                INCLUDE   "appvector.h"
;
;=====================================================================
;
        ORG     $FFD6
        dc      $01,$02,$03,$04,$05,$06,$07,$08
;
        ORG     VECTORS
;
KBD_INT         dc.w      JMP_KBD_ISR              ; Keyboard
SCI_TX_INT      dc.w      JMP_SCI_TX_ISR           ; SCI transmit
SCI_RX_INI      dc.w      JMP_SCI_RX_ISR           ; SCI receive
SCI_ERR_INT     dc.w      JMP_SCI_ERR_ISR          ; SCI error
T2OF_INT        dc.w      JMP_T2OF_ISR             ; TIM2 overflow
T2CH01_INT      dc.w      JMP_T2CH01_ISR           ; TIM Ch2_01
T2CH1_INT       dc.w      JMP_T2CH1_ISR            ; TIM Ch2_1
T2CH0_INT       dc.w      JMP_T2CH0_ISR            ; TIM Ch2_0
T1OF_INT        dc.w      JMP_T1OF_ISR             ; TIM1 overflow
T1CH01_INT      dc.w      JMP_T1CH01_ISR           ; TIM Ch1_01
T1CH1_INT       dc.w      JMP_T1CH1_ISR            ; TIM Ch1_1
T1CH0_INT       dc.w      JMP_T1CH0_ISR            ; TIM Ch1_0
IRQ1_INT        dc.w      JMP_IRQ_ISR              ; IRQ
USB_INT         dc.w      JMP_USB_ISR              ; USB device
SWI_INT         dc.w      JMP_SWI_ISR              ; SWI
;RST_IRQ        dc.w      JMP_Reset_Init           ; Reset vector
;
;
PROG_END:
        END
```

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

MOTOROLA

AN2399/D
Rev. 0
3/2003

Freescale Semiconductor, Inc.