

AN1742

Programming the 68HC705J1A In-Circuit

By Chris Falk
CSG Product Engineering
Austin, Texas

Introduction

This application note describes how a user can program the 68HC705J1A in-circuit. Programming in this way may be necessary when sections of code, such as lookup tables or calibration values, need to be programmed after the device is in-circuit.

Overview

The low-cost 68HC705J1A microcontroller unit (MCU) does not have a built-in function that allows in-circuit programming. The code included in [Appendix C](#) is similar to the bootloader code that is implemented on many MCUs. This bootloader code allows the MCU to receive data from a host computer and store this data in the EPROM. It must be pointed out that this is not a true in-circuit programming solution, because this solution requires that the code in [Appendix C](#) be programmed into the EPROM using an MCU programmer before the device is placed in the circuit.

Current production versions of the 68HC705J1A do not support programming the mask option register (MOR) in-circuit. It is not

advisable to program the MOR in-circuit because the contents of the MOR cannot be verified correctly until after the device has been RESET. Therefore, any desired mask options, excluding the security option, need to be programmed into the device before it is placed in-circuit.

The circuit shown in [Appendix B](#) must be added to the users end circuit also. The bootloader code uses this circuit to apply programming voltage to the MCU, convert the transmitted data from RS-232 levels to useful logic levels, and flash an LED (light-emitting diode) to alert the user to the programming status.

Preparing for Download

Once the bootloader code has been programmed into the device, the MCU is ready to begin downloading user code. Because the device is expecting data transmitted in Freescale S-record format, the user must compile the desired code in this format before transmission. An example S-record is shown in [Figure 1](#).

```
S1130400A6FFB705A608B701CD0311A80CB70120B5
S1130410F73FC0AE323CC026FC5A27043FC020F54C
S9040000FC
```

Figure 1. S-Record Example

The S1 indicates that this line has valid information, the 13 indicates the number of bytes in this line, and the 0400 is the address of the first byte in this record. The remaining information, up to the B5, is the opcodes and operands to be programmed. B5 is the checksum of the line, which is calculated by summing all of the opcodes and operands in that line and taking the complement. Each line of the S-record has the same format except the last line. The S9 on the last line terminates the S-record. The remaining information on the last line depends on compilers. In this case, this information indicates the starting address of the code.

Software Description

Execution begins by checking the status of location \$03E9. If this byte is programmed, the program counter jumps to the value of the variable USRCD. The user must enter the desired starting address in the variable USRCD. If location \$03E9 is blank, then the device will begin execution of the bootloader code. The device goes through an initializing sequence

in which it turns off the LED and applies programming voltage to the IRQ pin. It is necessary to apply programming voltage after the device is out of RESET. If programming voltage is applied before the RESET line is brought high, the part may come up in an unpredictable state.

Creating an SCI

The 68HC705J1A does not have a dedicated serial communications interface (SCI), so one must be created through software. This SCI is designed to work with an 8-bit data transmission with a start and stop bit. The code is timed to sample data being transmitted at a rate of 1200 bps. A diagram representing the incoming data stream is shown in **Figure 2**.

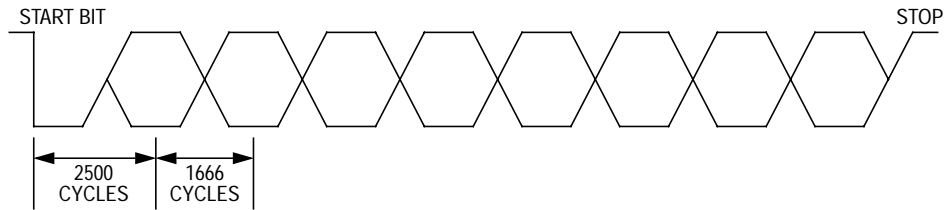


Figure 2. Transmitted Data

At a baud rate of 1200 bps, each bit takes 1/1200 second or 833.3 μ s to transmit. The software's timing is set up for a 4-Mhz external frequency, which gives an internal cycle time of 0.5 μ s. Each bit takes 1666.6 cycles to transmit based on this cycle time. PA0 is polled until the falling edge of the start bit is detected. The software then waits 2500 cycles before sampling the first bit. That number of cycles is approximately one and one half bit lengths, so the sample is taken midway through the transmission of the first bit. The remaining bits are sampled every 1666 cycles. The accumulator register acts as the receive register. Each received bit is shifted into the accumulator until the stop bit is detected.

NOTE: *No error detection techniques are built into this SCI. The user may add such features if memory space permits.*

Application Note

**Converting
ASCII Data
to Hexadecimal
Data**

The information contained in the accumulator is an ASCII character that must be converted to hexadecimal format to be programmed. Two ASCII characters are used to form one hex byte. The ASCII-to-hex conversion table is shown in [Table 1](#).

Table 1. ASCII to Hex Conversion Table

		First Hex Digit (MSB)							
		0	1	2	3	4	5	6	7
Second Hex Digit (LSB)	0	NUL	DLE	SP	0	@	P		p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BEL	ETB	'	7	G	W	g	w
	8	BS	CAN	(8	H	X	h	x
	9	HT	EM)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[k	{
	C	FF	FS	,	<	L	/		/
	D	CR	GS	-	=	M]	m	}
	E	SO	RS	.	>	N	^	n	
	F	SI	US	/	?	O	_	o	DEL

Here is an example to illustrate how ASCII data is converted to hex.

The byte to be programmed is \$A6. The host computer transmits the data for the ASCII character "A" (\$41 from the table). First, determine whether this is greater than or equal to \$41. If the character is greater

than or equal to \$41, subtract \$07 from it then AND the remainder with \$0F. If the character is less than \$41, simply AND the data with \$0F.

$$\$41 - \$07 = \$3A$$

$$\$3A \cdot \$0F = \$0A$$

The host computer then transmits the data for the ASCII character "6" (\$36 from the table). This is converted to hex using the second step from the above algorithm, since this data is less than \$41.

$$\$36 \cdot \$0F = \$06$$

Now that each character is converted to hex, they must be combined to form the original hex byte by multiplying the first character by \$10 and adding this value to the second character.

$$\$0A \times \$10 = \$A0$$

$$\$A0 + \$06 = \$A6$$

The data is now in a form that can be interpreted by the MCU.

Programming the EPROM

The data can be programmed to the EPROM using these steps:

1. Set the ELAT bit in the EPROG register.
2. Write desired value to desired location.
3. Set EPGM bit in EPROG register.
4. Wait time, t_{epgm} .
5. Clear EPGM and ELAT bits in EPROG register.

Setting the ELAT bit in the EPROG register causes the data and the address to be latched for programming, so it is not possible to execute code out of the EPROM while trying to program the EPROM. Therefore, the above routine must be moved into RAM and then executed.

After programming a byte, a verification step is performed to ensure that the byte was programmed properly. If for some reason the byte fails to verify, the programming voltage is removed from IRQ and the LED is turned off. It may be necessary to reprogram the device if this condition

occurs. If the byte does verify, the LED is toggled to indicate that the device is still operating properly.

This program and verify sequence continues until the "S9" is encountered in the S-record.

Location \$03E9 is then programmed to cause the MCU to execute user code upon the next reset.

NOTE: *Location \$03E9 is extremely important to bootloader execution. The bootloader code will be executed only if this location is blank. If this location is not blank, execution will begin at the location specified by the variable USRCD.*

Conclusion

This pseudo bootloader code offers the user a means to receive information serially and to program the MCU after it has been placed in-circuit. These concepts of software SCI and EPROM programming can be applied to other 68HC05 devices not offering such features in firmware.

Appendix A

Freescale Semiconductor, Inc.

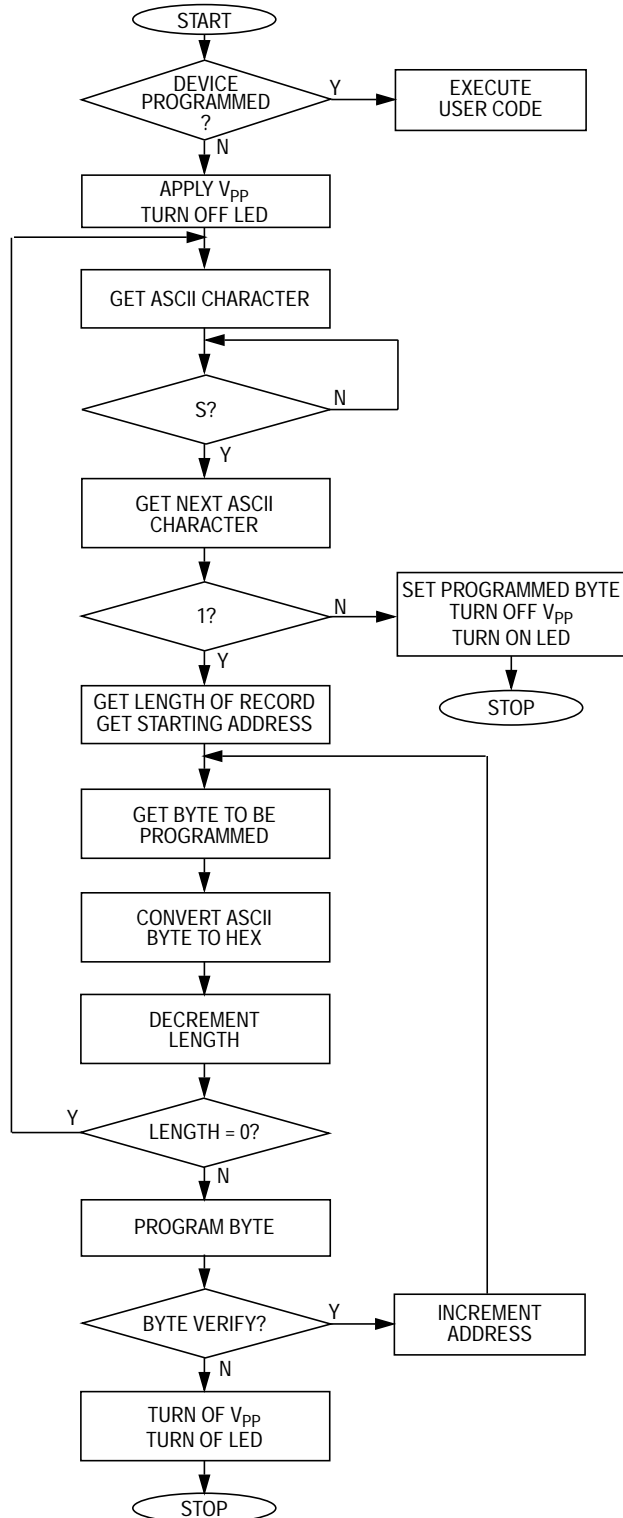


Figure 3. Pseudo Bootloader Code Flow

Appendix B

Freescale Semiconductor, Inc.

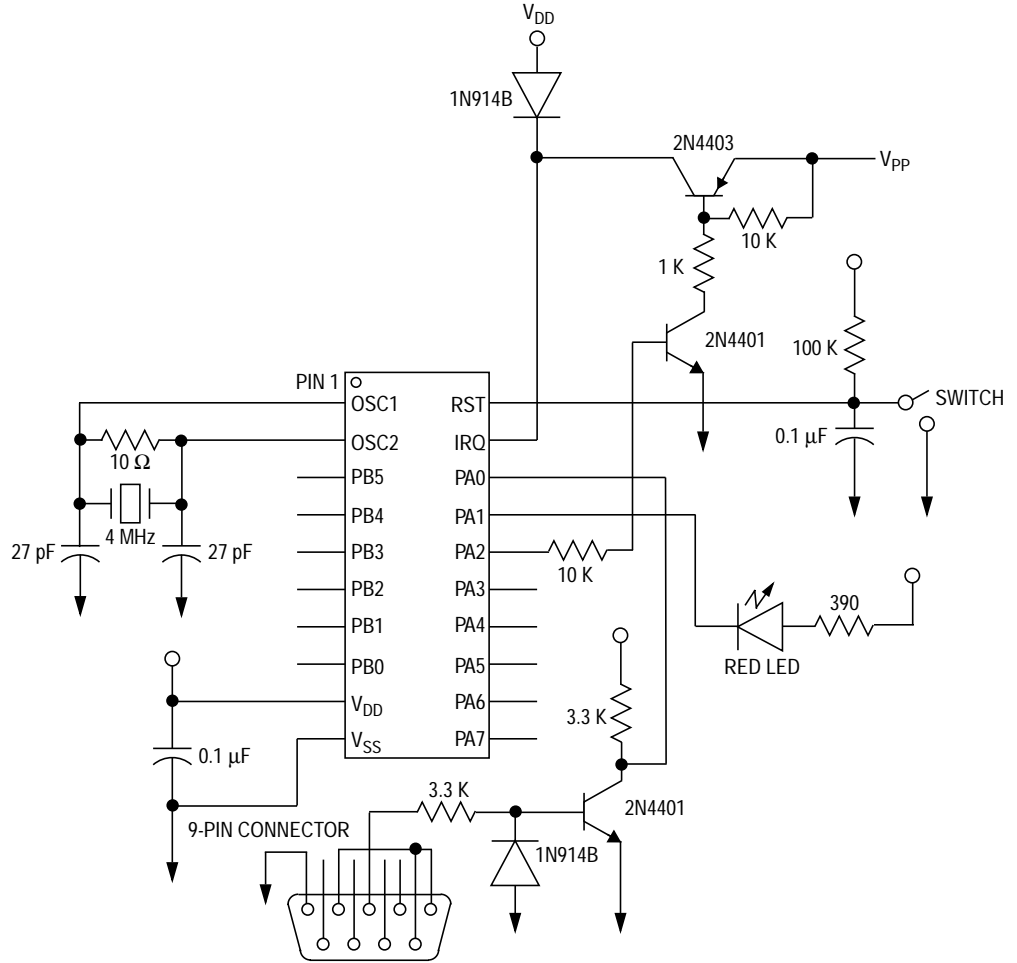


Figure 4. Circuitry Required for In-Circuit Programming

Appendix C

```

PORTA    EQU    $00
DDRA     EQU    $04
EPROG    EQU    $18
PRGSUB   EQU    $C0
VRFSUB   EQU    $D3
PRBYTE   EQU    $C3
IADDH    EQU    $C5
IADDL    EQU    $C6
LENGTH   EQU    $EA
USRCD    EQU    $0400
PRGFLG   EQU    $03E9
          ORG    $300

MAIN
          LDA    PRGFLG                ;IF LAST BYTE OF PRG CODE IS BLANK, PART
          BEQ    ICP                    ; HAS NOT BEEN PROGRAMMED.
          JMP    USRCD                  ;START DOWNLOADING.
          ;IF NOT, EXECUTE CODE

ICP
          LDA    #$06                    ;APPLY VPP AND TURN OFF
          STA    DDRA                    ;"FINISHED" LED
          STA    PORTA
          LDX    #$2A                    ;CONTAINS # OF BYTES TO MOVE TO RAM

MV2RAM
          LDA    PRGRT-1,X              ;MOVE PROGRAM AND VERIFY
          STA    PRGSUB-1,X            ;ROUTINES TO RAM
          DECX
          BNE    MV2RAM                  ;ALL MOVED?

SLOAD
          BSR    SCIRX                  ;GET FIRST CHARACTER
          CMP    #'S'                    ;IS IT S?
          BNE    SLOAD                  ;NO, WAIT FOR S
          BSR    SCIRX                  ;YES, GET NEXT CHARACTER
          CMP    #'1'                    ;IS IT 1?
          BNE    DONE                    ;NO, S RECORD IS FINISHED
          BSR    RCVASC                  ;YES, GET LENGTH OF RECORD
          SUB    #$02                    ;SUBTRACT ADDRESS BYTES
          STA    LENGTH                  ;STORE IT FOR LATER USE
          BSR    RCVASC                  ;GET UPPER ADDRESS
          STA    IADDH                    ;OF RECORD START
          BSR    RCVASC                  ;GET LOWER ADDRESS
          STA    IADDL                    ;OF RECORD START
          BRA    LPSTRT                  ;GO!

SLOOP
          LDA    #$02                    ;TOGGLE LED
          EOR    PORTA
          STA    PORTA
          LDA    IADDL                    ;IS ADDRESS TO BE
          CMP    #$F1                    ;PROGRAMMED $7F1(MOR)?
          BNE    NOTMOR                  ;YES, SET MPGM INSTEAD OF
          LDA    IADDH                    ;EPGM IN PRGSUB
          CMP    #$07                    ;NO, CONTINUE AS NORMAL
          BNE    NOTMOR
          JMP    SKIPMOR

NOTMOR
    
```

AN1742

Application Note

```

        JSR    PRGSUB                ;GOTO PROGRAMMING SUBROUTINE IN RAM
        BCLR   1,$C7
        LDA    IADDH                 ;MOVE ADDRESS TO VERIFY
        STA    $D6                   ;SUBROUTINE IN RAM
        LDA    IADDL
        STA    $D7
        JSR    VRFSUB                ;VERIFY THAT BYTE PROGRAMMED CORRECTLY
SKIPMOR
                                ;SKIP, WE DONT WANT TO PROGRAM THE MOR
                                ;MOVE TO NEXT ADDRESS
        JSR    INCADDR
LPSTRT
        JSR    RCVASC                ;GET NEXT BYTE
                                ;(TAKES INTO ACCOUNT CHKSM |)
        DEC    LENGTH                ;S-RECORD FINISHED?
        BNE    SLOOP                 ;NO, PROGRAM BYTE
        BRA    SLOAD                 ;YES, GET NEXT S-RECORD
DONE
        LDA    #$03
        STA    PRBYTE
        STA    IADDH
        LDA    #$E9
        STA    IADDL
        JSR    PRGSUB
        CLR    PORTA                 ;TURN OFF VPP, TURN ON FINISHED LED
        BRA    *
SCIRX
        BRSET  0,PORTA,*             ;WAIT FOR START BIT
        BSR    DLY378                ;DELAY 2500 CYCLES TO BE IN MIDDLE
        BSR    DLY378                ;OF TRANSMITTED BIT (1200 BAUD)
        BSR    DLY42
        CLRA
                                ;OF TRANSMITTED BIT
        SEC
                                ;CARRY IS USED AS STOP BIT
RX1
        BCS    RX2
        BRSET  0,PORTA,RX2          ;BURN A COUPLE CYCLES
                                ;BRSET SETS/CLEARs CARRY
RX2
                                ;DEPENDING ON EVALUATION
        LDX    #$04
WAIT
        BSR    DLY378                ;THIS LOOP BURNS 1560 CYLES
        DECX
                                ;TOTAL CYCLES BETWEEN BITS
        BNE    WAIT                 ;IS 1672
        BSR    DLY90
        RORA
                                ;MOVE CARRY BIT DOWN ACCUMULATOR
        BCC    RX1
                                ;REPEAT UNTIL STOP BIT REACHES CARRY
        BRCLR  0,PORTA,FRMERR
FRMERR
        RTS
                                ;DONE WITH THAT ASCII BYTE
RCVASC
        BSR    GETASC                ;GET NIBBLE OF BYTE
        BCS    GOTCR                 ;SKIP IF CONTROL CHAR
        BSR    SHIFT4                ;MOVE LOWER NIBBLE TO UPPER
        STA    PRBYTE                ;NIBBLE AND STORE FOR LATER USE
        BSR    GETASC                ;GET THE OTHER NIBBLE
        BCS    GOTCR                 ;SKIP IF CONTROL CHAR
        ORA    PRBYTE                ;COMBINE LOWER NIBBLE WITH
        STA    PRBYTE
        RTS
                                ;UPPER NIBBLE TO MAKE A BYTE

```

```

GETASC      BSR      SCIRX          ;GO GET A CHARACTER
            CMP      #'0'          ;LESS THAN 0?
            BLO     GOTCR          ;YES, ITS A CONTROL CHAR
            CMP      #'A'          ;NO, LESS THAN A?
            BLO     ONENINE        ;YES, CONTINUE AS NORMAL
            SUB     #$07          ;NO, CONVERT TO HEX

ONENINE     AND     #$0F          ;MASK OFF UPPER NIBBLE
            CLC

GOTCR      RTS

SHIFT4     LDX     #$10          ;MOVE LOWER NIBBLE
            MUL     ;TO UPPER NIBBLE
            RTS

INCADDR    INC     IADDL          ;INCREMENT LOWER ADDRESS
            BNE     RETURN        ;EQUAL TO 00? NO, RETURN
            INC     IADDH          ;YES, BUMP HIGH ADDRESS

RETURN     RTS

PRGRT     BSET     2,EPROG        ;SET ELAT
            LDA     #PRBYTE        ;STORE DESIRED BYTE
            STA     $0400          ;TO DESIRED LOCATION
            BSET     0,EPROG        ;SET EPGM
            LDX     #$03          ;DELAY 3X378 CYCLES

DLYLP     BSR     DLY378
            DECX
            BNE     DLYLP
            CLR     EPROG          ;CLEAR ELAT AND EPGM
            RTS          ;DONE

VRFRT     LDA     PRBYTE          ;COMPARE VALUE TO BE PROGRAMMED
            CMP     $0400          ;WITH ACTUAL PROGRAMMED VALUE
            BEQ     PASS          ;ARE THEY =?
            BSET     1,PORTA        ;NO TURN OFF LED

FAIL      BRA     *              ;AND HANG

PASS      RTS          ;YES, GO ON

DLY378    BSR     DLY186
DLY186    BSR     DLY90
DLY90     BSR     DLY42
DLY42     BSR     DLY18
DLY18     BSR     DLY6
DLY6      RTS

RESET     ORG     $7FE
            DW     $0300 _
    
```

AN1742

Application Note

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 Halbleiter 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.

