MOTOROLA

TECHNICAL UPDATE

MC68HC705C8
MC68HC705C8A
MC68HSC705C8A
MC68HC705C4A

Technical Update contains updates to documented information appearing in other Motorola technical documents as well as new information not covered elsewhere.

We are confident that your Motorola product will satisfy your design needs. This Technical Update and the accompanying manuals and reference documentation are designed to be helpful, informative, and easy to use.

Should your application generate a question or a problem not covered in the current documentation, please call your local Motorola distributor or sales office. Technical experts at these locations are eager to help you make the best use of your Motorola product. As appropriate, these experts will coordinate with their counterparts in the factory to answer your questions or solve your problems. To obtain the latest document, call your local Motorola sales office.

TABLE OF CONTENTS

TECHNICAL UPDATE

Modules

Serial Peripheral Interface (SPI) Module

SPI_A

Revision History

Date	Revision	Description
5/2/95	**2.00**	Includes trackers HC705C8.004, HC05C8.005, and HC05C8.006.

SPI Test Program

Reference Document: Not applicable

Tracker Number: HC705C8.004 Revision: 2.00

The following code will work on all MCUs that have the SPI_A module. The code was tested on an HC705C8. Some equates and vectors may have to be changed to properly work on a specific part.

```
************************************************************
*
* Program Name: 7C8_SPI.ASM ( SPI Test on the 705C8 )
* Revision: 1.00
* Date: June 7, 1993
*
* Written By: Mark Glenewinkel
*             Motorola CSIC Applications
*
* Assembled Under: P&E Microcomputer Systems IASM05
*
*       *********************************
*       *       Revision History        *
*       *********************************
*
*       Rev     1.00    06/07/93        Mark Glenewinkel
*                       Initial Release
*
*****************************************************************
```

```
*
* Program Description:
*
*       Use the HC705C8 resident MCU on the HC05EVM to
*         run this test.
*       Jumper pin #34 on header J19 on the EVM to 5v through
*         A 10kOhm resistor. This ties the SS pin of the SPI
*         high insuring against the possibility of a mode
*         fault error.
*       Download the program.
*       Make sure the PC is at $800.
*       Type GO.
*       Look at pin #32 of header J19. This is the MOSI pin
*         of the SPI. You should see '$55' come out of this
*         pin. The MOSI pin's steady state level is a logic '1'.
*         The bitstream's width is 8usecs. Each bit using
*         1usec of time. The program executes in an
*             infinite loop.
*       ABORT the program to stop operation.
*
*****************************************************************

***     Equates for 705C8
SPCR    EQU     $0A                     ;spi ctrl reg
SPSR    EQU     $0B                     ;spi status reg
SPDR    EQU     $0C                     ;spi data reg


***     Start of program                ***

        ORG     $0800                   ;start of user eprom

START   LDA     #$50
        STA     SPCR                    ;spi enabled, mstr
                                        ;  cpha=cpol=spr1=spr0=0

AGAIN   LDA     #$55
        STA     SPDR                    ;send $55 out on spi

LOOP    LDA     SPSR                    ;load spi status reg
        AND     #$80                    ;check if SPIF bit is set
        BEQ     LOOP                    ;if not, go back
                                        ;  and check again

        BRA     AGAIN
```

SPI Code Snippet (Master)

Reference Document: HC68HC05C4/D, Section 6

Tracker Number: HC05C8.005 **Revision: 1.00**

The following code will work on all MCUs that have the SPI_A module. The code was tested on an HC705C8. Some equates and vectors may have to be changed to properly work on a specific part.

```
*******************************************************************
*
* Program Name: C8SPIM.ASM
* Revision:     2.0
* Target MCU:   MC68HC05C8, MC68HC705C8
* Date:         8/18/93
* Written by:   David Yoder, Motorola CSIC Applications
* Assembly:     P&E Microcomputer Systems IASM05 3.0m
*
*******************************************************************
* Rev history:
* Rev 2.0       8/18/93         No longer outputs through DACIA of
*                               EVM. Only outputs PA2 if error occurs.
*
* Rev 1.0       8/13/93         original
*******************************************************************
*
* This code shows a basic SPI transfer protocol between one master
* and one slave. A string is continuously transmitted to the slave.
* The companion slave program reverses the case of all alpha
* characters before sending the message back. The message received
* by the master is again case switched and then compared to the
* original message. If a difference is noted, PA2 is driven low. PA2
* idles high.
*
* In order for the handshaking to operate, the MCU's should be
* connected as shown below.
*
* Master                  Slave
* ------                  -----
* PD2/MISO -------------- PD2/MISO
* PD3/MOSI -------------- PD3/MOSI
* PD4/SCK  -------------- PD4/SCK
* PD5/SS -----\
* PA1 --------/
* PA0 ------------------- PD5/SS
* PA2 ------------------O Error indicator
*
* PA0 controls SS_ (slave select) on the slave. For the mode used
* (cpol=1, cpha=0), the slave SS_ must be brought high between each
* transfer. If it is not, a write collision will result when the
* slave SPDR is written.
*
* PA1 controls SS_ on the master.
*  MC68HCx05C8: The master SS_ must be pulled high while the SPI is
*   enabled in master mode. If it is not, a mode fault will result.
*  MC68HCx05C9: The master PD5/SS pin may be set as output in DDRD bit5.
*   If this is done, master SS_ need not be pulled high. This was not
*   done to insure compatibility with the MC68HCx05C8, which has an
*   input-only Port D.
*
* PA2 pulses low to indicate transmission errors.
*
* PortD DDR
*  MC68HCx05C8: Does not have a data direction register. No need to
*   write to address $07.
*  MC68HCx05C9: Has data direction register. It must be set up
*   appropriately for the SPI to operate.
**********************************************************************

************** Equates *******************
cr      equ     $0d     ;Carriage Return character
lf      equ     $0a     ;Line Feed character
```

```
************ MCU Equates *****************
porta   equ     $00
ddra    equ     $04
portd   equ     $03
ddrd    equ     $07

ROM0    equ     $20     ;Start of ROM0
RAM     equ     $50     ;Start of main RAM

************ SPI Equates *****************
spcr    equ     $0a     ;SPI control register
spsr    equ     $0b     ;SPI status register
spdr    equ     $0c     ;SPI data register

************* SPI Bit Equates ************

********** SPCR ***********
spie    equ     7       ;SPI interrupt enable bit
spe     equ     6       ;SPI enable
mstr    equ     4       ;SPI master enable
cpol    equ     3       ;SPI clock polarity
cpha    equ     2       ;SPI clock phase
spr1    equ     1       ;SPI rate
spr0    equ     0       ;SPI rate

********** SPSR ************
spif    equ     7       ;SPI interrupt flag
wcol    equ     6       ;SPI write collision
modf    equ     4       ;SPI mode fault

********** PortA ***********
sss     equ     0       ;port a0 is tied to ss on slave spi
mss     equ     1       ;port a1 is tied to ss on the master
error   equ     2       ;port a2 pulses low when an SPI error
                        ; is caught

***************DACIA Equates*******************

IER     equ     $20     ;interrupt enable register(write)
ISR     equ     $20     ;interrupt status register(read)
TDR     equ     $23     ;transmit data register(write)
RDR     equ     $23     ;receive data register(read)

*************DACIA bit Equates********************

DTDRE   equ     6       ;transmit data reg. empty

***************End of Equates******************

************** Variables **********************
        org     RAM     ;start of main RAM
temp    rmb     1       ;temporary variable
```

```
************** Reset Vectors ******************
        org     $1ff4           ;vectors
SPI     fdb     trap
SCI     fdb     trap
TIMER   fdb     trap
IRQ     fdb     trap
SWI     fdb     trap
RESET   fdb     start


        org     $0200           ;start of program area

****************Program Beginning**************

start:  bsr     spistrsetup     ;initialise system
        bsr     checksetup
start10 ldx     #msg            ;point to string
start20 bsr     spistr          ;xmit one char of string
        beq     start10         ;start over if end of string
        bsr     casesw          ;reverse case of rec'd char
        bsr     check           ;same as xmit'd char?
        bra     start20         ;xmit next char

**********************************************
* Subroutine: spistrsetup
* Inputs:
*       none
* Outputs:
*       none
* Alters Regs:
*       A
*       CCR
*
**********************************************
spistrsetup:
        lda     #$18            ;sck=1,mosi=1 this does nothing on C8
        sta     ddrd            ;but MUST be done on C9

        lda     #{1<sss + 1<mss}
                                ;left shift 1's into these bit positions
                                ;port a0 controls ss on the slave spi
                                ;port a1 controls ss on the master
        sta     ddra

        bset    sss,porta       ;deselect slave
                                ; the slave ss must go low during the
                                ; transfer and high between transfers
                                ; for the mode cpha=1,cpol=0. If the
                                ; spdr of the slave is written while
                                ; ss of the slave is low, a write
                                ; collision will occur.
        bset    mss,porta       ;deselect master
                                ; the master ss must be held high during
                                ; all time that the master spe and mstr
                                ; bits are set. A mode fault will result
                                ; if it is not held high during this
                                ; time.
        lda     #{1<spe + 1<mstr + 1<cpol + 1<spr0}
                                ;left shift 1's into these bit positions
                                ;set the master up as follows
                                ; do not enable spi interrupts
                                ; enable spi
                                ; enable master mode
                                ; cpol=1
                                ; cpha=0
                                ; spr=01 : sck=eck/4
        sta     spcr

        rts                     ;return from subroutine

**********************************************
* Subroutine: checksetup
* Inputs:
*       none
* Outputs:
*       none
* Alters Regs:
*       none
*
**********************************************
checksetup:
        bset    error,ddra      ;set error bit as output
        bset    error,porta     ;put error flag in idle state
                                ; pa2 will toggle low if an error is
                                ; detected in the SPI system

        rts                     ;return from subroutine
********************SPI Data Transfer***********
* Subroutine: spistr
* Inputs:
*       X: address of string to xmit
* Outputs:
*       A:      character received by SPI
*       X:      address of next character to xmit
*       CCR:    Z bit set if end of string is reached
* Depends upon:
*       spistrsetup
* Alters Regs:
*       A
*       X
*       CCR
*       Variable TEMP
* Description:
* Transmits one character of a string out the SPI
* system. Returns with Z bit set when the "$"
* is reached. Returns with Z bit clear if "$"
* is not reached.
************************************************

spistr:
        lda     ,x              ;get message data
        cmp     #"$"            ;is it the end of the message?
        beq     spistr10        ;done with string
                                ;return with Z bit set
        bclr    sss,porta       ;select slave
        sta     spdr            ;send character
        brclr   spif,spsr,*     ;check spif, wait until set
        bset    sss,porta       ;deselect slave
                                ; slave must be deselected so that
                                ; it can write to it's own spdr
                                ; and not cause a write collision
        incx                    ;set up to send next byte

        lda     spdr            ;get the recieved character

        clr     temp            ;we are not done with the
        com     temp            ; string, so insure that Z
                                ; bit is clear for return

spistr10:
       rts                      ;return to calling routine
```

```
*******************************************************
* Subroutine: casesw
* Intputs:
*       ASCII character in acc
* Outputs:
*       ASCII character in acc
* Alters Regs:
*       A
*       CCR
*
* Routine changes upper case to lower case and
* lower case to upper case. Leaves non-alpha
* characters unchanged.
********************************************************

casesw  cmp     #$41            ;below alphas?
        bmi     casesw20
        cmp     #$5b            ;above caps?
        bpl     casesw10
        add     #$20            ;must be cap, change to low
        bra     casesw20
casesw10:
        cmp     #$61            ;between alphas?
        bmi     casesw20
        cmp     #$7b            ;above alphas?
        bpl     casesw20
        sub     #$20            ;must be lowercase, change to cap

casesw20:
        rts                     ;return
*******************************************************
* Subroutine: check
* Intputs:
*       A: value to check
*       X: pointer to next character to xmit
*          offset from received char by -2
* Outputs:
*       A: unaffected
*       X: unaffected
* Depends upon:
*       checksetup
* Alters Regs:
*       CCR
*
* Routine compares the value in accumulator to value
* pointed to by (X-2).
*
* If the value do not match, PA0 is pulsed low
*
* If X=msg+1, the routine returns immediately.
*
* This is usefull for comparing
* received to transmitted data with the SPI.
********************************************************

check:
        cpx     #msg+1          ;was this the 1st xfer?
        beq     check20         ;if so, don't bother
        decx                    ;X=X-2
        decx
        cmp     ,x              ;rec char = xmit char?
        beq     check10
        bclr    error,porta     ;if not, pulse error
        bset    error,porta
```

```
check10:
        incx                ;X=X+2
        incx
check20:
        rts                 ;done

****************Trap************************

trap    bra     trap            ;trap for unused vectors

**************Message data******************
        org     ROM0            ;store message in Page0 ROM
msg:    db      "The Quick Brown Fox jumped over the Lazy Dog",cr,lf,"$"
```

SPI Code Snippet (Slave)

Reference Document: HC68HC05C4/D, Section 6

Tracker Number: HC05C8.006 Revision: 2.00

The following code will work on all MCUs that have the SPI_A module. The code was tested on an HC705C8. Some equates and vectors may have to be changed to properly work on a specific part.

```
*******************************************************************
*
* Program Name: C8SPIS.ASM
* Revision:     1.1
* Target MCU:   MC68HC05C8, MC68HC705C8
* Date:         8/18/93
* Written by:   David Yoder, Motorola CSIC Applications
* Assembly:     P&E Microcomputer Systems IASM05 3.0m
*
*******************************************************************
* Rev History:
* 1.1           8/18/93         changed label names for consistancy
* 1.0           8/12/93         original for MC68HCx05C9 memory map
*******************************************************************
*
* This code shows a basic SPI transfer protocol between one master
* and one slave. This slave module receives characters, changes
* the case of all alpha characters, and transmits the character
* back. Non-alpha characters are transmitted unchanged.
*
* In order for the handshaking to operate, the master should use the
* code snippet (C9SPIM.ASM), and the MCU's should be connected as
* shown below.
*
* Master                  Slave
* ------                  -----
* PD2/MISO -------------- PD2/MISO
* PD3/MOSI -------------- PD3/MOSI
* PD4/SCK  -------------- PD4/SCK
* PD5/SS -----\
* PA1 --------/
* PA0 ------------------- PD5/SS
* PA2 ------------------O Error indicator
*
```

```
* PA0 in the master code snippet controls SS_ (slave select) on the
* slave. For the mode used (cpol=1, cpha=0), the slave SS_ must be
* brought high between each transfer. If it is not, a write collision
* will result when the slave SPDR is written.
*
* PA1 controls SS_ on the master. The master code snippet is written
* such that the master controls it's own slave select line.
*  MC68HCx05C8: The master SS_ must be pulled high while the SPI is
*   enabled in master mode. If it is not, a mode fault will result.
*  MC68HCx05C9: The master PD5/SS pin may be set as output in DDRD bit5.
*   If this is done, master SS_ need not be pulled high. This was not
*   done to insure compatibility with the MC68HCx05C8, which has an
*   input-only Port D.
*
* PA2 of the master code snippet pulses low to indicate transmission
* errors.
*
* PortD DDR
*  MC68HCx05C8: Does not have a data direction register. No need to
*   write to address $07.
*  MC68HCx05C9: Has data direction register. It must be set up
*   appropriately for the SPI to operate.
**********************************************************************

************** Equates *******************
cr      equ     $0d             ;carriage return character
lf      equ     $0a             ;line feed character
dc1     equ     $11
************ SPI Equates *****************

portd   equ     $03             ;port d
ddrd    equ     $07             ;data direction register for port d
spcr    equ     $0a             ;SPI control register
spsr    equ     $0b             ;SPI status register
spdr    equ     $0c             ;SPI data register

************* SPI Bit Equates ************

******* SPCR ***********
spie    equ     7               ;SPI interrupt enable bit
spe     equ     6               ;SPI enable bit
mstr    equ     4               ;SPI master mode bit
cpol    equ     3               ;SPI clock polarity bit
cpha    equ     2               ;SPI clock phase bit
spr1    equ     1               ;SPI rate bit 1
spr0    equ     0               ;SPI rate bit 0

******* SPSR ************
spif    equ     7               ;SPI interrupt flag bit
wcol    equ     6               ;SPI write collision bit
modf    equ     4               ;SPI mode fault bit

***************End of Equates******************

        org     $1ff4           ;reset vectors

*************** Vectors ************************

SPI     fdb     echosw
SCI     fdb     trap
TIMER   fdb     trap
IRQ     fdb     trap
SWI     fdb     trap
reset   fdb     start

        org     $0200           ;start of program area
```

```
****************Program Beginning**************

start:
        bsr     setup
        cli                     ;enable system wide interrupts

start10:
        nop                     ;wait for interrupts
        bra     start10

*****************Init SPI*********************
* Subroutine: setup
* Inputs:
*       none
* Outputs:
*       none
*
* Initializes SPI system***********************************************
setup:  lda     #$04            ;set up PD2/MOSI as output
        sta     ddrd            ;others as input
                                ;MUST be done on C9,
                                ;has no effect on C8

        lda     #{1<spie + 1<spe + 1<cpol + 1<spr0}
                                ;shift 1's into appropriate
                                ; bit postions
        sta     spcr            ;setup SPI as follows:
                                ; enable SPI interrupts
                                ; enable SPI system
                                ; do not enable master mode
                                ; cpol=1 : in this mode, ss must
                                ; cpha=0 : go high between xfers
                                ; spr=01 : sck=eck/4
        rts

********************SPI Data Transfer ISR*******
* ISR: echosw
* Depends upon:
*       setup
*       casesw
*
* Slave SPI ISR.
* Receives character from SPI system. Assumes the character
* to be ASCII. Switches the case of all alpha characters.
* Does not affect non-alpha characters. Transmits the
* resulting character back to master.
***************************************************
echosw:
        brclr   spif,spsr,*     ;make sure transmission is complete

        lda     spdr            ;get data received from SPI

        bsr     casesw          ;reverse the case of alphas

echosw10:
        sta     spdr            ;send data
                                ; with cpha=1, ss must go low
                                ; before this write is made.
                                ; If not, a write collision
                                ; will occur. In this example,
                                ; the master controls the slave
                                ; ss line.
        brset   wcol,spsr,echosw10
                                ;if a write collision occurred,
                                ;try again

        rti                     ;return to main loop

*******************************************************
* Subroutine: casesw
* Intputs:
*       ASCII character in acc
* Outputs:
*       ASCII character in acc
* Alters Regs:
*       A
*       CCR
*
* Routine changes upper case to lower case and
* lower case to upper case. Leaves non-alpha
* characters unchanged.
********************************************************

casesw  cmp     #$41            ;below alphas?
        bmi     casesw20
        cmp     #$5b            ;above caps?
        bpl     casesw10
        add     #$20            ;must be cap, change to low
        bra     casesw20
casesw10:
        cmp     #$61            ;between alphas?
        bmi     casesw20
        cmp     #$7b            ;above alphas?
        bpl     casesw20
        sub     #$20            ;must be lowercase, change to cap

casesw20:
        rts                     ;return

*******************Trap************************
trap    bra     *               ;trap for unused vectors
```

Computer Operating Properly (COP) COP0COP

Revision History

Date	Revision	Description
6/27/95	1.00	Includes tracker HC705P6.012

COP Timeout Period

Reference Documents: MC68HC705C8AD/D Rev. 4.0, pages 14 (705C4A), 31 (705C8A), and 51; HC705C5GRS/D Rev. 1.3, page 49; HC05P1AGRS/D Rev 1.3; MC68HC05P4/D, page 4-2; HC05P5GRS/D Rev. 1.3; MC68HC05P7/D, page 4-2; HC05P15GRS/D Rev. 0.0, page 33; HC05P18GRS/D Rev. 0.5, page 12.

Tracker Number: HC705P6.012 Revision: 1.00

The timeout period for the watchdog timer on the COP0COP is a direct function of the crystal frequency. The equation is:

$$\text{Timeout Period} = \frac{262{,}144}{\text{Fxtal}}$$

For example, the timeout period for a 4-MHz crystal would be 65.536 ms.

CPU

HC05CPU

Revision History

Date	Revision	Description
5/3/95	1.00	Includes trackers HC05CPU.001, HC705C8.002R2, HC705C8.017, HC705C8.018R2, and HC705C8019.

Correction to SUB in Applications Guide

Reference Documents: M68HC05 Applications Guide MC68HC05AG/AD, page A-62; M68HC05 Applications Guide MC68HC05AG/AD Rev. 1, page A-62

Tracker Number: HC05CPU.001 Revision: 1.00

Replace the C bit description with:

The C bit (carry flag) in the condition code register gets set if the absolute value of the contents of memory is larger than the absolute value of the accumulator, cleared otherwise.

External Interrupt Timing

Reference Documents: MC68HC705C8/D Rev. 1, page 3-5; MC68HC05B6/D, Rev. 3, page 11-11, note 4; MC68HC705C8/D, Rev. 1, page 3-5; MC68HC05C9/D, page 13-7, note 3; MC68HC05C12/D, page 13-9, note 4; MC68HC05D9/D, Rev. 1, page 10-4, note 1; MC68HC05J3/D, page 9-6, note 3; and MC68HC05X16/D, page 12-6, note 4

Tracker Number: HC705C8.002 Revision: 2.00

This time (Tilil) is obtained by adding 19 instruction cycles to the total number of cycles needed to complete the service routine. The return to interrupt (RTI) is included in the 19 cycles.

I Bit in CCR During Stop Mode

Reference Document: M68HC05 Applications Guide, page 3-93

Tracker Number: HC705C8.017 Revision: 1.00

The stop mode flow chart shows that the I bit is set when stop mode is entered. However, this is not true. The I bit actually is cleared when stop mode is entered so that an external IRQ may release the processor from stop mode.

This error is present in the original applications guide as well as the revision.

BSET and BCLR are Read-Modify-Write Instructions

Reference Documents: MC68HC705C8/D Rev. 1, page 7-6; MC68HC05J1/D Rev. 1, page 5-7; MC68HC05J3/D, page 8-4; MC68HC705J2/D, page 4-16; HC05J3/705J3 Technical Databook - MC68HC05J3/D, page 8-6; MC68HC05K1/D, page 10-10; MC68HC705K1/D, page 11-10

Tracker Number: HC705C8.018 Revision: 2.00

In many data books, the read-modify-write instruction table located in the instruction set and addressing mode section does not list the BSET and BCLR instructions. These data books list BSET and BCLR as bit-manipulation instructions only.

While this is correct, it is not complete. These operations use a read-modify-write method to accomplish their task and, therefore, should be included in the table of read-modify-write instructions.

NOTE: These instructions do not use the same addressing modes as the other read-modify-write istructions. Only direct addressing is valid for BSET and BCLR.

Because BSET and BCLR are read-modify-write instructions, they may not be used with write-only registers. These registers will read back undefined data. Therefore, a read-modify-write operation will read undefined data, modify it as appropriate, and then write it back to the register. Because the original data is undefined, the data written back will be undefined also.

I Bit in CCR During Wait Mode

Reference Document: M68HC05 Applications Guide, page 3-93

Tracker Number: HC705C8.019 Revision: 1.00

The wait mode flow chart does not show that the I bit gets cleared upon entering wait mode. The I bit is cleared when wait is entered. An external IRQ or any of the internal interrupts (timer, SCI, SPI) can release the processor from wait mode.

This error is present in the original applications guide as well as the revision.

Timer Module

TIM1IC1OC_A

Revision History

Date	Revision	Description
5/3/95	2.00	Includes trackers HC05C4.002, HC05C4.003, and HC705P9.005.

Input Capture/Output Compare Code Snippet

Reference Document: Not applicable

Tracker Number: HC05C4.002 Revision: 2.00

Changes include added memory map disclaimer.

```
****************************************************************
*
*       Program Name: ICOCC4.ASM
*       Revision: 1.0
*       Date: 9/6/93
*
*       Written By: Mark Johnson
*                   Motorola CSIC Applications
*
*       Assembled Under: P&E Microcomputer Systems
*                        IASM05 Version 3.02m
*
*                 ********************************
*                 *       Revision History       *
*                 ********************************
*
*       Revision 1.00   9/1/93 Original Release
*
****************************************************************
*
*       Program Description:
*
*       This was written for the timer module TIM1IC1OC_A and tested
*         on the HC05C4. In order to use this with other HC05 MCU's,
*         reset vectors and memory map equates may have to be changed.
*         See the Technical Databook for the appropriate part for this
*         memory map information.
*
*       This simple program was written to demonstrate the input
*       capture and output compare functions of the MC68HC(8)05C4
*       timer. The routine generates a level transition on port A
*       which is fed into the input capture pin (TCAP). When
*       the input capture occurs an offset of 50us is added to
*       value in the input capture registers and stored in the
*       output compare registers. The output compare generates
*       a level transition on the TCMP pin and then the entire
*       process is repeated.
*
```

```
*
*       The program was run on the M68HC05EVM using the
*       following setup conditions:
*
*       1) HC705C8 Resident Processor
*       2) Fop = 2MHz
*       3) Pin 11 (PA0) on target header J19 jumpered to pin
*          37 (TCAP).
*       4) The user should see a level transition on the
*          TCMP pin approximately* 50us after the level
*          transition on port A.
*
*   *NOTE: The level transition on the TCMP pin will occur at
*          50us + 1 count of the free-running counter = 52us.
*          This is the result of an internal synchronization
*          delay which occurs during an input capture.
*          ( 1 count = 4 internal bus cycles)
****************************************************************
*
*       Register Equates
*
porta           equ     $00             ;port A data register
ddra            equ     $04             ;port A data dir. reg.
tcr             equ     $12             ;timer control register
tsr             equ     $13             ;timer status register
inpcaph         equ     $14             ;input capture (MSB)
inpcapl         equ     $15             ;input capture (LSB)
outcomph        equ     $16             ;output compare (MSB)
outcompl        equ     $17             ;output compare (LSB)
*
*      RAM Variables
*
                org     $50             ;RAM address space
templ           rmb     1               ;storage for O/C low byte
*
*      Beginning of main routine
*
                org     $200            ;EPROM/ROM address space
start           lda     #$ff
                sta     ddra            ;all port A pins are outputs
                clra
                sta     porta           ;output a low on port A
                lda     #3
                sta     tcr             ;IEDG = positive edge
                                        ;OLVL = high output
loop            lda     tsr             ;read timer status register
                lda     outcompl        ;clear OCF
                com     porta           ;toggle port A
                lda     #!25            ;I/C low byte offset
                add     inpcapl         ;add I/C low byte value
                sta     templ           ;save new value in temp storage
                lda     inpcaph         ;get high byte of I/C reg.
                adc     #0              ;add carry from last addition
                sta     outcomph        ;store value to O/C high byte
                lda     templ           ;get low byte offset
                sta     outcompl        ;store value in O/C low byte
                lda     inpcapl         ;enable input captures
                brclr   6,tsr,*         ;wait for output compare
                lda     tcr             ;get Timer Control Register
                eor     #3              ;toggle IEDG and OLVL
                sta     tcr             ;store new IEDG and OLVL values
                bra     loop            ;repeat process indefinitely
*
*      Reset vector setup
*
                org     $1ffe
                fdb     start
```

Interrupt Driven Output Compare Code

Reference Document: MC68HC05C4/D (ADI-991-R2), page 4-7

Tracker Number: HC05C4.003 Revision: 2.00

Changes include added memory map disclaimer.

The following code listing shows the procedure of using the output compare function driven by an interrupt to produce a square wave. The code was tested with an HC705C8 on the HC05EVM board. The code will work on an HC05C4.

```
*****************************************************************
*
* Program Name: 7C8_OCI.ASM ( Square wave generation on OC )
* Revision: 1.00
* Date: September 29, 1993
*
* Written By: Mark Glenewinkel
*             Motorola CSIC Applications
*
* Assembled Under: P&E Microcomputer Systems IASM05
*
*       ********************************
*       *       Revision History       *
*       ********************************
*
*       Rev     1.00    09/29/93        Mark Glenewinkel
*                       Initial Release
*
*****************************************************************
*
* Program Description:
*
*       This was written for the timer module TIM1IC1OC_A and tested
*         on the HC05C4. In order to use this with other HC05 MCU's,
*         reset vectors and memory map equates may have to be changed.
*         See the Technical Databook for the appropriate part for this
*         memory map information.
*
*       This program uses the Output Compare function of the
*         timer to generate a square wave. The output compare
*         interrupt is utilized to take care of adding the
*         appropriate value to the 16 bit output compare
*         register to create the square wave. With some
*         modification, this routine can perform pulse width
*         modulation.
*
*       Use the HC705C8 resident MCU on the HC05EVM to
*         run this test.
*       Download the program.
*       Make sure the PC is at $1000. Type GO.
*       OR, hit USER RESET on the EVM.
*       Look at pin #35 of header J19. This is the Timer
*         Compare Output pin (TCMP) of the timer. You should
*         see a 3.906kHz square wave on this pin with a
*         256 usec period.
*       Press ABORT on the EVM to halt program execution.
*
*****************************************************************
```

```
***     Equates for 705C8
TCR     equ     $12                     ;timer ctrl reg
TSR     equ     $13                     ;timer status reg
OCH     equ     $16                     ;output compare high reg
OCL     equ     $17                     ;output compare low reg
TCH     equ     $18                     ;timer counter high reg
TCL     equ     $19                     ;timer counter low reg
TEMP    equ     $50                     ;temp loc for OCL

***     Start of program                ***

        org     $1000                   ;start of user code
START   lda     #$41                    ;output compare interrupt
                                        ;enabled, output level 0
        sta     TCR                     ;store to timer ctrl reg
        cli                             ;clear the I bit in CCR

DUMLOOP bra     DUMLOOP                 ;dummy loop waiting for
                                        ; timer interrupt


***     Interrupt Service Routine       ***
OCISR   lda     TSR                     ;read timer status
                                        ; to clear flag

*       Flip the OLVL bit in the TCR reg
        lda     TCR                     ;load ACCA w/ TCR
        eor     #$01                    ;flip bit 0 of ACCA
        sta     TCR                     ;store ACCA to TCR

*       Add 64 counts to timer counter reg
*       With a 2 MHz internal bus clock, the timer count
*         period is 2 usec. 64 counts of the timer counter
*         will produce a square wave half cycle of 128 usecs.
        lda     #$40                    ;load #$40 into acca
        add     OCL                     ;add OCL to ACCA
        sta     TEMP                    ;store res to temp loc
        lda     #$00                    ;add $00 to out comp hi
        adc     OCH                     ; with carry
        sta     OCH                     ;store res to out comp hi
        lda     TEMP                    ;store temp to out
        sta     OCL                     ; comp low

        rti                             ;return from interrupt

***     Set up vectors
        org     $1FF8                   ;define timer
        dw      OCISR                   ; interrupt vector

        org     $1FFE                   ;define reset vector
        dw      START
```

Input Capture Test

Reference Document: Not applicable

Tracker Number: HC705P9.005 **Revision: 2.00**

Listed below is a program that tests the input capture function on the HC705P9 on the HC05P9EVS.

```
*****************************************************************
*
* Program Name: P9_INCAP.ASM ( Input Capture Test for the P9EVS )
* Revision: 1.00
* Date: June 7, 1993
*
* Written By: Mark Glenewinkel
*             Motorola CSIC Applications
*
* Assembled Under: P&E Microcomputer Systems IASM05
*
*       *********************************
*       *       Revision History        *
*       *********************************
*
*       Rev     1.00    06/07/93        M.R. Glenewinkel
*                       Initial Release
*
*****************************************************************
*
* Program Description:
*
*       This was written for the timer module TIM1IC1OC_A and tested
*         on the HC705P9. In order to use this with other HC05 MCU's,
*         reset vectors and memory map equates may have to be changed.
*         See the Technical Databook for the appropriate part for this
*         memory map information.
*
*       Tests the Input capture pin.
*       Use the HC705P9 resident MCU on the HC05P9EVS to
*         run this test.
*       Jumper pins PA0 and PD7/TCAP on Target Header P4.
*       We will use Port A, bit 0 to toggle the TCAP pin.
*       Download the program.
*       Make sure the PC is at $100.
*       Type GO.
*       ABORT the program and look at locations $80-$83.
*         After the first Input Capture, the Input Capture
*         Registers High and Low are loaded into RAM
*         location $80 and $81, respectively. After the
*         second Input Capture, the Input Capture Registers
*         High and Low are loaded into RAM location $82
*         and $83, respectively.
*       If you trace this program, the Input capture
*         flag will look like its not being set when you
*         view with the emulator software. Remember, the
*         flag gets cleared when a read of ICL and TSR occurs.
*         The emulator software does this automatically when
*         reading those locations to display in the
*         emulator window.
*
*****************************************************************
```

```
***     Equates
PORTA   EQU     $00
PORTB   EQU     $01
PORTC   EQU     $02
DDRA    EQU     $04
DDRB    EQU     $05
DDRC    EQU     $06
DDRD    EQU     $07
TCR     EQU     $12
TSR     EQU     $13
ICRH    EQU     $14
ICRL    EQU     $15

TEMP1   EQU     $0080
TEMP2   EQU     $0081
TEMP3   EQU     $0082
TEMP4   EQU     $0083


***     Start of code

        ORG     $0100                   ;start of program

START   LDA     #$FF
        STA     PORTA                   ;PortA is $FF
        LDA     #$00
        STA     DDRD                    ;PortD is input
        LDA     #$FF
        STA     DDRA                    ;PortA is output
        STA     DDRC

        LDA     #$00
        STA     TCR                     ;set InCap to fall edge
        LDA     TSR                     ;look at tsr
        LDA     ICRL                    ;look at input reg low
                                        ;this clears any flags

        LDA     #$00                    ;falling edge created
        STA     PORTA                   ; on PortD/TCAP

LOOP    LDA     TSR                     ;wait in loop for flag
        AND     #$80                    ; to be set
        BEQ     LOOP

        LDA     ICRH                    ;write counter values
        STA     TEMP1                   ;in memory
        LDA     ICRL
        STA     TEMP2

        LDA     #$02                    ;set InCap to rising edge
        STA     TCR
        LDA     #$FF                    ;rising edge created
        STA     PORTA                   ;on PortD/TCAP

LOOP2   LDA     TSR                     ;wait in loop for flag
        AND     #$80                    ; to be set
        BEQ     LOOP2

        LDA     ICRH                    ;write counter values
        STA     TEMP3                   ;in memory
        LDA     ICRL
        STA     TEMP4

LOOP3   NOP
        BRA     LOOP3
```

Parts Specific

MC68HC705C8

SPIF Bit Errata

Revision History

Date	Revision	Description
5/2/95	1.00	Initial release. Includes trackers HC705C8.001, HC705C8.003, HC705C8.006, HC705C8.015, HC705C8.020, and HC705C8.021.

Reference Document: HC68HC705C8/D Rev. 1, page 6-7

Tracker Number: HC705C8.001 Revision: 1.00

Errata statement:

When the serial peripheral interrupt (SPI) is operating as a slave and the serial clock (SCK) is asynchronous to the E clock, SPI transfers can fail. The failure rate is in the range of one error in several million transfers. The error is expressed differently for CPHA = 0 vs. CPHA = 1, but the cause is always due to metastable behavior of an internal flip-flop whose clock is related to E and whose data input is related to the SPI SCK signal. (Errors occur when the data input is in transition at exactly the same time as the flip-flop clock changes.)

When CPHA = 0, the error is expressed as the failure of data to transfer from the shift register to the parallel receive data buffer (SPDR) at the end of a transfer. When CPHA = 1, the error is expressed as a failure of SPIF to be set at the end of a transfer. Also, when CPHA = 1, the data transfer error described for CPHA = 0 can occur, although even less often than the SPIF error.

Because they are rare and difficult to produce, most users would attribute these errors to random system noise. A hardware logic fix has been identified and is included on new mask sets, which currently are being qualified.

Users are encouraged to design their software to tolerate rare transfer errors, such as those caused by this bug or by random noise. Remember that the SPI transfer protocol does not tolerate noise on the SCK clock.

No mask sets of MC68HC705C8 have the fix.

New part MC68HC705C8A, which is in qualification, contains the fix.

Keypad Decoding

Reference Document: M68HC05AG/AD, pages 4-18 through 4-21

Tracker Number: HC705C8.003 Revision: 1.00

This is a general clarification of the keypad routine used on the thermostat project described in the guide.

Clarification:

Section 4 of the *M68HC05 Applications Guide*, especially the keypad checkout program, should be studied along with this document. While studying this document, refer to the figure of a typical keypad schematic at the end of the text.

Assume that the MCU port is port B and that it is configured such that bits 4 through 7 are inputs and bits 0 through 3 are outputs.

Also assume that the output bits are initialized to all ones and are attached to the keypad columns. The input bits are attached to the keypad rows. The rows also have pulldown resisters.

With everything in a static state as described above, a read of the input/output (I/O) port would return a value of 00001111 or $0F.

Initially, the program will loop to read the port and check if the upper nibble has changed from 0000 to any other value. If the port has not changed, the program continues looping. For example, if the push button labeled zero is pushed, the upper nibble would read 1000.

Next, turn off all but one of the column outputs. Then read the port to ensure that ones are on the row inputs. If the ones are there, turn off the current column and turn on the next one and repeat until a value greater than zero is read from the row inputs. If a zero value is read after scanning all columns, the key was released.

The table below contains all of the possible values that can be generated by the keypad. This “look-up table” can be used automatically to scan the columns and do the comparisons. A code example also is included to help demonstrate what is happening.

Key #		Row 7654	Column 3210	Hex Value
key:	0	1000	1000	$88
	1	1000	0100	$84
	2	1000	0010	$82
	3	1000	0001	$81
	4	0100	1000	$48
	5	0100	0100	$44
	6	0100	0010	$42
	7	0100	0001	$41
	8	0010	1000	$28
	9	0010	0100	$24
	A	0010	0010	$22
	B	0010	0001	$21
	C	0001	1000	$18
	D	0001	0100	$14
	E	0001	0010	$12
	F	0001	0001	$11
null:		0000	0000	$00

The first two instructions make the column outputs all ones. This has to be done before every scan so that it can be determined if a key has been pressed.

```
lda       $0F                          ;make col outputs all ones
sta       portb                        ;someplace else portb lower nibble was
                                       ;made an output
```

At this point, the port is checked repeatedly until a one is read on one of the row inputs. If a zero is read, no key has been pressed. Because the interest is only in the row input bits, the lower nibble is ignored.

```
first:    lda       portb               ;key data from keypad port
and       #$F0                          ;we are interested in only the row inputs
                                        ;so mask off the cols.
beq       first                         ;if no key is pushed go back and check
                                        ;again.
```

Once a key is pressed, determine which key it is. To do so, output all possible key combinations to port B, read the data back, and compare it to the output. This is easily done using a table to generate all of the possible values. The index register is used to increment through the table. The index register should be initialized to zero or to the total number of entries into the table. If the latter is used when the index register is zero, the end of the table has been reached. In this example, the end of table byte (null character) will be checked to see if that value has been accessed.

```
ldx       #$00                         ;init the index register
keylp:    bnlda     key,x              ;This loads acca with the value point to by key +x
                                       ;the first time thru it would be$88
cmp       $00                          ;check to see if end of table
beq       end                          ;must be end of table
sta       portb                        ;write to portb.This will affect only the lower 4 bits
```

Next, the port is immediately read back. For example, if the first table entry is pointed to, in this case $88, this value is driven out of the port. Remember, only the col bits are driven, and they will have a value of 1000. If the zero key was pushed, bit 7 of the port would be high because the column that is being driven intersects with the row the zero key is on. If the port is read at this time, 1000 1000 will be returned, the same value that is being driven out. If the two key had been pushed, a value of $08 would have been returned because the column that intersects with the row is being driven with a zero. So comparing the value driven out by the value being returned shows if the right entry is in the table. The following instructions indicate how this is done.

```
cmp       portb                       ;acca has the value driven out. The cmp instruction read
                                      ;reads the port and does the compare at the same
                                      ;time
beq       found                       ;we've found the right entry
```

Organize the table a little differently than the table above so that the value to be driven out of the port and the ascii value this code represents results. This is useful when the right location in the table is found. The contents of the index register and the starting address of the table (key) added together point to the correct entry. Adding one to the value of the index register points to the ascii equivalent of the key being pressed.

The first byte in each entry is the value generated for each column and row pattern. The second byte of each pair is what that combination should represent.

key: fcb	$88,'0'	;1000 1000
fcb	$84,'1'	;1000 0100
fcb	$82,'2'	;1000 0010
fcb	$81,'3'	;1000 0001
fcb	$48,'4'	;0100 1000
fcb	$44,'5'	;0100 0100
fcb	$42,'6'	;0100 0010
fcb	$41,'7'	;0100 0001
fcb	$28,'8'	;0010 1000
fcb	$24,'9'	;0010 0100
fcb	$22,'A'	;0010 0010
fcb	$21,'B'	;0010 0001
fcb	$18,'C'	;0001 1000
fcb	$14,'D'	;0001 0100
fcb	$12,'E'	;0001 0010
fcb	11,'F'	;0001 0001
fcb	$00	;0000 0000 null character to show end of table

If the right pattern is not found, the index register is incremented twice to skip over the current pattern and the next byte, which is the ascii equivalent of that pattern. The new value pointed to is loaded into acca and checked to see if it is the end of table character. If not, the new pattern is output to the port and the routine starts from the beginning until the correct pattern is found or the end of table is reached. When the correct key is found, incrementing the index register by one points to the ascii equivalent byte.

```
incx
incx
bra           keylp                    ;start the search again.

found:                                 ;this is where we go to if correct pattern
                                       ;is found.  We can increment the index register
incx                                   ;and retrieve the ascii equivalent if we want
                                       ;when done we can start the scan over again.
end:                                   ;do what we want exit start over or error recovery
```

Also consider keypad debounce. After a key press is detected, a debounce delay should be inserted and the key checked again after the delay period.

Mask Set C16W Bootloader Code

Reference Document: Not applicable

Tracker Number: HC705C8.006 Revision: 1.00

```
***********************************************************
*                                                         *
*             68HC705C8 EPROM BOOTLOADER PROGRAM          *
*             ==================================          *
*                                                         *
*              This version:- 10/11/89 - REV 6            *
*              FIXED 'PROGRAM SECURITY' MODE              *
*                                                         *
***********************************************************
*
* I/O DEFINITIONS
*
PORTA   EQU     $00     PORT A DATA
PORTB   EQU     $01     PORT B DATA
PORTC   EQU     $02     PORT C DATA
PORTD   EQU     $03     PORT D DATA   (Input Only!)
DDRA    EQU     $04     PORT A DDR
DDRB    EQU     $05     PORT B DDR
DDRC    EQU     $06     PORT C DDR
*
* SERIAL PERIPHERAL INTERFACE REGISTERS
*
SPCR    EQU     $0A     SERIAL PERIPHERAL CONTROL
SPSR    EQU     $0B     SERIAL PERIPHERAL STATUS
SPDAT   EQU     $0C     SERIAL PERIPHERAL DATA
*
* SERIAL COMMUNICATIONS INTERFACE REGISTERS
*
BAUD    EQU     $0D     BAUD RATE CONTROL
SCCR1   EQU     $0E     SERIAL COMM'S CONTROL REGISTER 1
SCCR2   EQU     $0F     SERIAL COMM'S CONTROL REGISTER 2
SCSR    EQU     $10     SERIAL COMM'S STATUS
SCDAT   EQU     $11     SERIAL COMM'S DATA
*
* TIMER REGISTERS
*
TIMCR   EQU     $12     TIMER CONTROL
TIMSR   EQU     $13     TIMER STATUS
IPCAPH  EQU     $14     INPUT CAPTURE  (High Byte)
IPCAPL  EQU     $15     INPUT CAPTURE  (Low  Byte)
OPCOMH  EQU     $16     OUTPUT COMPARE (High Byte)
OPCOML  EQU     $17     OUTPUT COMPARE (Low  Byte)
COUNTH  EQU     $18     COUNTER (High Byte)
COUNTL  EQU     $19     COUNTER (Low  Byte)
DUALTH  EQU     $1A     DUAL TM REGISTER (High Byte)
DUALTL  EQU     $1B     DUAL TM REGISTER (Low  Byte)
*
* EPROM CONTROL REGISTER
*
PROG    EQU     $1C     EPROM CONTROL
*
```

```
* MEMORY MAP DEFINITIONS
*
EPROM0  EQU     $20     BASE OF PAGE ZERO EPROM AREA
RAM     EQU     $50     BEGINNING OF RAM
STACK   EQU     $FF     STACK RESET ADDRESS
EPROM   EQU     $0100   BASE OF MAIN EPROM AREA
BOOTST  EQU     $1F00   START OF BOOTSTRAP ROM AREA
BOOTV   EQU     $1FE0   START OF BOOTSTRAP VECTOR AREA
VECTOR  EQU     $1FF0   START OF USER VECTOR AREA
*
* RAM VARIABLES
*
RAMSUB  EQU     RAM        LOCATION OF RAM SUBROUTINE
ADDR    EQU     RAMSUB+1   EXTENDED ADDRESS FOR RAM SUBROUTINE
LOOP    EQU     RAMSUB+4   INTER-BYTE DELAY
TIME    EQU     RAMSUB+5   PROGRAMMING PULSE WIDTH
*
* PORT A DEFINITIONS
*
ADDRLO  EQU     PORTA      LOW  ORDER ROM ADDRESSES A0 - A7
A5      EQU     5          BIT 5 :- ADDRESS LINE A5
*
* PORT B DEFINITIONS
*
DATAIN  EQU     PORTB      ROM DATA INPUT PORT
*
* PORT C DEFINITIONS
*
ADDRHI  EQU     PORTC      HIGH ORDER ROM ADDRESSES A8 - A12
VFYLED  EQU     5          BIT 5 DRIVES 'VERIFY' LED
PRGLED  EQU     6          BIT 6 DRIVES 'PROGRAMMING' LED
TSC     EQU     7          BIT 7 CONTROLS EXTERNAL ROM TRI-STATING..
*                          OUTPUT PIN - 0=MEMORY ENABLED  1= MEMORY
*                          TRI-STATED (FLOAT IF NOT USED).
*
* PORT D DEFINITIONS
*
MODES   EQU     PORTD      BOOTSTRAP MODE INPUT PORT
PIND2   EQU     2          BIT 2
PIND3   EQU     3          BIT 3
PIND4   EQU     4          BIT 4
PIND5   EQU     5          BIT 5
*
* MISCELLANEOUS DEFINITIONS
*
CMASK   EQU     %11100000  PORTC CONTROL LINES MASK
EPGM    EQU     0          PROG  BIT0; - Vpp CONTROL BIT
ERASED  EQU     $00        VALUE OF AN ERASED EPROM BYTE
INSTAT  EQU     %01100000  INITIAL PORT C LED STATUS
LAT     EQU     2          PROG  BIT2; - EPROM ADDRESS LATCH BIT
LATCH   EQU     %00000100  PROG  BIT2
MUL     EQU     $42        OP-CODE FOR MULTIPLY INSTRUCTION
OCF     EQU     6          TIMSR BIT6; - OUTPUT COMPARE FLAG
OLVL    EQU     0          TIMCR BIT0; - TIMER COMPARE OUTPUT LEVEL
RDRF    EQU     5          SCSR  BIT5; - RCV DATA REG FULL FLAG
TDRE    EQU     7          SCSR  BIT7; - XMIT DATA REG EMPTY FLAG
TEST    EQU     2          PORTD BIT2; - '0' GO BOOT,'1'GO $51 (RAM)
OPTION  EQU     $1FDF      OPTION REGISTER
TSTREG  EQU     $1F        TEST REGISTER
*
```

```
*******************************************************************
*
*    INITIAL REGISTER VALUES
*       FCB     %00100000   PORT A :- ROM ADDRESSES A7-A0 SET TO $20
*       FCB     $00         PORT B :- DON'T CARE
*       FCB     %01100000   PORT C :- ROM ENABLED/ LED'S OFF.....
*                               ...A12-A8 ZEROED
*       FCB     $00         PORT D :- DON'T CARE
*       FCB     %11111111   PORT A DDR :- ALL OUTPUTS
*       FCB     $00         PORT B DDR :- ALL INPUTS
*       FCB     %11111111   PORT C DDR :- ALL OUTPUTS
*       FCB     $00         SPCR   :- DISABLE OPERATION
*       FCB     $00         SPSR   :- DON'T CARE
*       FCB     $00         SPDAT  :- DON'T CARE
*       FCB     %00110000   BAUD   :- =4800Baud @2Mhz XTAL
*       FCB     %00000000   SCCR1  :- 8 DATA BITS
*       FCB     %00001100   SCCR2  :- INHIBIT INTERRUPTS, ENABLE...
*                                     RX/TX NO WAKE-UP, NO BREAK
*       FCB     $00         SCSR   :- DON'T CARE
*       FCB     $00         SCDAT  :- DON'T CARE
*       FCB     $00         TIMCR  :- DISABLE TIMER INTERRUPTS
*       FCB     $00         TIMSR  :- DON'T CARE
*       FCB     $00         IPCAPH :- DON'T CARE
*       FCB     $00         IPCAPL :- DON'T CARE
*       FCB     $00         OPCOMH :- DON'T CARE
*       FCB     $00         OPCOML :- DON'T CARE
*       FCB     $00         COUNTH :- DON'T CARE
*       FCB     $00         COUNTL :- DON'T CARE
*       FCB     $00         DUALTH :- DON'T CARE
*       FCB     $00         DUALTL :- DON'T CARE
*       FCB     $00         PROG   :- DISABLE PROGRAMMING
*
*
*   RAM AREA IS INITIALISED AS FOLLOWS;
*
*   LOCATION:-            INSTRUCTION:-
*
* RAMSUB  $50     $C7     STA    EPROM0
* ADDR    $51     $00
* ADDR+1  $52     $20
*         $53     $81     RTS
*
*   TIMER VARIABLES
*
* LOOP    $54       2          -
* TIME    $55       1          -
*
*
*****************************************************************
        ORG     BOOTST
*
TABLE   FCB     $C7             'STA EXTENDED' INSTRUCTION
        FCB     $00             ADDRESS $0020
        FCB     $20             .
        FCB     $81             'RTS' INSTRUCTION
        FCB     2               2 PASSES THRU PROG
        FCB     $01             1 mS PROGRAMMING PULSE CONSTANT
*
START   EQU     *
*
* FIRST CHECK FOR SECURITY AND HANG IF ENABLED
*
        LDA     OPTION
        AND     #%00001000
        BEQ     NOSEC
        STOP
*
```

```
* THEN CHECK PORT D, BIT 2, TO SEE IF USER WISHES TO JUMP TO
* RAM OR JUMP INTO THE BOOTLOADER PROGRAM.
*
*
NOSEC   BRCLR   TEST,PORTD,BOOT
        JMP     RAM+1               GO TO RAM PROGRAM AT $0051
*
*
*
* SET UP PORTS, TIMER, RAM SUBROUTINE, AND RAM VARIABLES
*
*
BOOT    CLR     SCCR1          SET SCI TO 8 DATA BITS OPERATION
        LDX     #DDRA          POINT TO DDRA & INIT X FOR MULTIPLY
        COM     ,X             SET PORTA TO ALL OUTPUTS
        COM     DDRC           SET PORTC TO ALL OUTPUTS
*
        LDA     #%00001100     GET SCCR2 INITIAL VALUE
        STA     SCCR2          SET UP SCCR2
*
*       GET BAUD INITIAL VALUE (N.B. MUL NOT SUPPORTED BY ASSEMBLER!)
        FCB     MUL            X=0, A=%00110000 <== X=4, A=%00001100
        STA     BAUD           SET UP BAUD
*
        LSLA                   INITIAL PORT C VALUE (%01100000)
        STA     PORTC          ENABLE ROM, TURN OFF LED'S, ZERO..
*                              ZERO A12-A8
        LDA     #EPROM0        GET INITIAL PORT A (A7 - A0) VALUE
        STA     ,X             SET UP PORT A
*
*  INITIALISE RAM SUBROUTINE AND VARIABLES THEN JUST FILL RAM
*
MOVE    LDA     TABLE,X        GET A BYTE FROM THE TABLE
        STA     RAM,X          MOVE IT INTO RAM
        INCX                   POINT TO THE NEXT BYTE TO BE MOVED
        BNE     MOVE           KEEP MOVING UNTIL ALL ARE IN PLACE
*
*
*
        BRCLR   PIND5,MODES,MOVED
*
*DO THE GATE STRESS TEST
*
GTEST   LDA     #$15    MOE/TS1=0,STE/ST0=1 ALSO RCKH=1
        BRSET   PIND3,MODES,SEC
        STA     TSTREG  ENABLE GATE STRESS TEST
*                       ...AND OUTPUT XMIT CLOCK ON TDO
        BSET    0,ADDR  SET UP FOR ADDRESS $0120
        BRCLR   PIND4,MODES,NOCOM
        COM     ADDR+1  USE ALTERNATE ADDRESS ($01DF)
NOCOM   BSR     ZAP
        WAIT
*
*PROGRAM THE SECURITY BIT
*
SEC     LSLA            A HAS $15 FROM START OF GTEST
        BSET    LAT,PROG
        STA     OPTION  (A=$2A <== A=$15)
        BSR     ZAPSUB
*
```

```
**********************************************************************
*
* THE BOOTLOADER PROGRAM HAS 5 MODES OF OPERATION:
*
*   I. PROGRAM/VERIFY - PERFORMS A NORMAL PROGRAM CYCLE FOLLOWED BY A
*                       VERIFY CYCLE WHICH HANGS IF THE EPROM IS NOT
*                       CORRECTLY PROGRAMMED. EITHER 1ms OR 5ms PULSE
*                       WIDTH CAN BE SELECTED.
*
*  II. VERIFY - PERFORMS ONLY A VERIFY CYCLE WHICH HANGS IF THE EPROM
*               IS NOT CORRECTLY PROGRAMMED.
*
* III. LOAD RAM - LOADS A PROGRAM FROM SCI INTO THE RAM
*                 THEN JUMPS TO RAM TO EXECUTE THE PROGRAM.
*
*  IV. DUMP EPROM - DUMPS THE EPROM CONTENTS OF THE 705C8 TO THE SCI
*
*   V. SECURE THE PART - PROGRAMS ONLY THE SECURITY BIT AND THEN
*                        EITHER DOES ANOTHER VERIFY (NOT THE SEC BIT)
*                        OR HANGS IN THE DUMP EPROM MODE
*
* WHEN COMING OUT OF RESET INTO THE BOOTLOADER PROGRAM (ASSUMING THAT
* PORT D PIN 2 ALLOWS YOU TO ENTER THE BOOTLOADER) THE STATE OF
* PORT D PINS 3, 4 AND 5 DETERMINES WHICH MODE OF OPERATION THE
* PROGRAM WILL ENTER.
*
* THE GATE STRESS TEST CAN ALSO BE INVOKED THROUGH THE BOOTLOADER.
*
*    |--------------------------|---------------------|
*    |         PORT D           |                     |
*    |-------|--------|---------|       MODE          |
*    | PIN 5 | PIN 4  |  PIN 3  |                     |
*    |-------|--------|---------|-------------------------------|
*    |   0   |   0    |    0    |   PROGRAM/VERIFY  15ms PULSE  |
*    |-------|--------|---------|-------------------------------|
*    |   0   |   0    |    1    |   VERIFY                      |
*    |-------|--------|---------|-------------------------------|
*    |   0   |   1    |    0    |   LOAD RAM                    |
*    |-------|--------|---------|-------------------------------|
*    |   0   |   1    |    1    |   DUMP EPROM                  |
*    |=======|========|=========|===============================|
*    |   1   |   0    |    0    |   GATE STRESS TEST            |
*    |-------|--------|---------|-------------------------------|
*    |   1   |   0    |    1    |   SECURE/VERIFY               |
*    |-------|--------|---------|-------------------------------|
*    |   1   |   1    |    0    |   GATE STRESS TEST            |
*    |-------|--------|---------|-------------------------------|
*    |   1   |   1    |    1    |   SECURE/DUMP EPROM           |
*    |=======|========|=========|===============================|
*
**********************************************************************
*
*
* CHECK PORT D PINS 3, 4 AND 5 TO DETERMINE WHICH MODE TO ENTER
*
*
MOVED   BRCLR   PIND4,MODES,PRGVER      DO A PROGRAM/VERIFY OR VERIFY
        BRCLR   PIND3,MODES,LDRAM       DO A LOAD RAM - EXECUTE CYCLE
*
```

```
* DUMP THE CONTENTS OF THE 705C8 EPROM TO THE SCI OUTPUT.
*        ( ASSUMES 'RAMSUB' CONTAINS $C7 )
*
DMPEPR  DEC     RAMSUB          CHANGE 'STA' TO 'LDA' EXTENDED ($C6).
DUMPIT  JSR     RAMSUB          READ ONE BYTE OF EPROM
WAITTX  BRCLR   TDRE,SCSR,WAITTX  WAIT FOR TRANSMIT REGISTER TO EMPTY
        STA     SCDAT           SEND EPROM DATA TO SERIAL OUTPUT
        BSR     NXTADR          MOVE TO NEXT ADDRESS
        BNE     DUMPIT          IF NOT FINISHED, CONTINUE
        WAIT                    HANG WHEN FINISHED.
*
*
* CHOOSE BETWEEN PROGRAM/VERIFY AND JUST VERIFY MODES
*
*
PRGVER  BRSET   PIND3,MODES,VERIFY      DO A VERIFY CYCLE
        BCLR    PRGLED,PORTC            LIGHT 'PROGRAMMING' LED
*
*
* PROGRAM THE EPROM WITH THE CONTENTS OF THE EXTERNAL ROM
*
*
PRGLOP  BSR     PRGSUB                  PROGRAM ONE EPROM BYTE
        BSR     NXTADR                  POINT TO NEXT ADDRESS
        BNE     PRGLOP                  KEEP PROGRAMMING UNTIL DONE
        STA     ADDR+1                  RESET LOW ORDER  ADDR TO $20
        STA     PORTC                   RESET HIGH ORDER ADDRESS ON PORTC TO $20
        CLR     ADDR                    RESET HIGH ORDER ADDR TO $00
        BSET    A5,ADDRLO               PUT $20 ON PORT A
        DEC     LOOP                    DECREMENT LOOP COUNTER
        BNE     PRGLOP                  RELOAD PARAM. FOR 2ND PASS
*
*
* VERIFY THE EPROM CONTENTS AGAINST EXTERNAL MEMORY.
*        ( ASSUMES 'RAMSUB' CONTAINS $C7 )
*
VERIFY  LDA     #INSTAT         GET INITIAL LED STATUS FOR A VERIFY
COMPAR  INC     RAMSUB          CHANGE 'STA' TO 'EOR' EXTENDED ($C8).
        STA     PORTC           PLACE LED AND ROM STATUS ON PORT PINS
        BSET    A5,ADDRLO       SET A7 - A0 TO $20 (START OF EPROM)
*
CHECK   LDA     DATAIN          READ A BYTE FROM THE EXTERNAL MEMORY
        JSR     RAMSUB          COMPARE TO AN EPROM BYTE
        BNE     *               HANG IF THEY DON'T MATCH
        BSR     NXTADR          POINT TO NEXT ADDRESS TO BE COMPARED
        BNE     CHECK           KEEP CHECKING BYTES UNTIL EPROM END
*
DONE    BCLR    VFYLED,PORTC    INDICATE EPROM VERIFIED AS CORRECT
        STOP
*
*
*
* LOAD THE RAM WITH A USER'S PROGRAM VIA THE SCI.
*
*    THE DATA SHOULD BE IN THE FORM OF 1 START-BIT, 8 DATA-BITS,
* 1 STOP-BIT. THE FIRST BYTE SHOULD BE A COUNT OF THE TOTAL NUMBER
* OF BYTES TO BE SENT, INCLUDING THAT BYTE. THE FIRST BYTE IS
* LOADED INTO $50 SO THE FIRST PROGRAM BYTE WILL BE LOADED INTO
* $51. THAT IS WHERE PROGRAM EXECUTION WILL BEGIN.
*    IF A COUNT IS USED THAT IS GREATER THAN ( PROGRAM LENGTH + 1 )
* THEN THE ROUTINE WILL HANG AFTER THE LAST PROGRAM BYTE IS SENT.
* THIS CAN BE USED TO HOLD OFF EXECUTION OF THE PROGRAM UNTIL BIT-2
* OF PORTD IS SET AND RESET IS ASSERTED.
*
*
```

```
LDRAM   LDX     #RAM    POINT TO START OF RAM
WAITRX  BRCLR   RDRF,SCSR,WAITRX   WAIT FOR RX REGISTER TO FILL
        LDA     SCDAT   READ DATA BYTE FROM RX REGISTER
        STA     ,X      STORE THE DATA IN RAM
        INCX            MOVE TO NEXT RAM LOCATION
        DEC     RAM     DECREMENT THE PROGRAM SIZE COUNTER (1st BYTE)
        BNE     WAITRX  IF ENTIRE PROGRAM NOT LOADED, CONTINUE
JMPRAM  JMP     RAM+1   JUMP TO THE PROGRAM IN THE RAM
*
*******************************************************************
*                                                                 *
*                      S U B R O U T I N E S                      *
*                                                                 *
*******************************************************************
*
* PROGRAM AN EPROM ADDRESS WITH DATA RECEIVED FROM PORTB.
* THE ADDRESS TO BE PROGRAMMED SHOULD BE PLACED IN LOCATION
* 'ADDR' & 'ADDR+1'.
*
*
PRGSUB  LDA     DATAIN          READ DATA FROM PORTB
        BEQ     SKIP            RETURN IF EQUAL TO ERASED STATE ($00)
*
ZAPSUB  BSR     ZAP
        CLR     PROG           REMOVE Vpp FROM CIRCUIT
*                               CLEAR THE LAT BIT
SKIP    RTS
*
ZAP     BSET    LAT,PROG
        JSR     RAMSUB          WRITE ONE BYTE OF DATA
        BSET    EPGM,PROG      APPLY Vpp TO CIRCUIT
        LDA     TIME            GET PROGRAMMING PULSE LENGTH
*****************************************************************
*
*
* DELAY N mS SUBROUTINE. ON ENTRY, ACCUMULATOR SHOULD CONTAIN THE
* TIME DELAY WANTED IN MILLISECONDS. AT THE END OF THE DELAY BIT
* 'OLVL' WILL BE CLOCKED TO THE 'TCMP' PIN.
* ( ASSUMES 2MHz OSCILLATOR FREQUENCY ).
*
*
DELNMS  LDX     #$A6            1 MS INNER LOOP
MS1     DECX
        BNE     MS1
        DECA                    DECREMENT OUTER LOOP
        BNE     DELNMS
        RTS                     RETURN AFTER WANTED DELAY

        NOP                     PADDING TO KEEP REST OF CODE SAME
        NOP
        NOP
        NOP
        NOP
*
*******************************************************************
*
*  NXTADR SUBROUTINE
*
*  COMPUTES NEXT EPROM ADDRESS TO BE PROGRAMMED, VERIFIED, OR DUMPED.
*  PUTS THIS ADDRESS ON PORTS A (LS BYTE) AND C (MS BYTE) AND
*  UPDATES RAMSUB ALSO. SKIPS THE RAM, BOOTSTRAP AND UNUSED AREAS.
*  RETURNS WITH Z=1 IF THE COMPUTED ADDRESS IS = $2000, MEANING THAT
*  A PASS THROUGH THE MEMORY MAP HAS BEEN COMPLETED. OTHERWISE Z=0.
*
```

```
NXTADR  LDA     ADDR            GET MS.BYTE OF LAST ADDRESS USED
        BNE     NOT04F          BRANCH IF NOT IN PAGE ZERO EPROM AREA
        LDX     ADDR+1          GET LS.BYTE OF LAST ADDRESS USED
        CPX     #RAM-1          LAST BYTE OF PAGE ZERO EPROM?
        BNE     NOT04F          BRANCH IF NOT $004F
*
* MUST HAVE JUST ACCESSED LAST PAGE ZERO EPROM LOCATION IF HERE SO
* FORCE NEXT ADDRESS INCREMENT TO POINT TO MAIN EPROM AREA AT $0100.
*
        LDX     #$0100-1        POINT TO ADDRESS BELOW THE ONE WANTED
        STX     ADDR+1          PLACE IN LOCATION TO BE INCREMENTED
        STX     ADDRLO          PRESET PORTA PRIOR TO INCREMENT
*
* 16 BIT ADDRESS INCREMENT
*
NOT04F  INC     ADDR+1          INCREMENT LS. ADDRESS BYTE
        INC     ADDRLO          UPDATE LS.ADDRESS AT PORT A
        BNE     GOBACK          RETURN IF A PAGE BOUNDARY NOT REACHED
        INCA                    INCREMENT MS. ADDRESS
*
* LOOK OUT FOR HAVING GONE THROUGH THE ENTIRE MEMORY MAP.
*
        CMP     #$20            WAS THAT THE END OF MEMORY ( $1FFF )?
        BEQ     GOBACK          EXIT WITH Z=1 IF THE END WAS REACHED.
*
* LOOK OUT FOR HAVING ACCESSED THE LAST LOCATION IN THE MAIN BLOCK
*
        CMP     #$1F            WAS THAT THE END OF THE MAIN BLOCK
        BNE     INMAIN          BRANCH IF STILL WITHIN THE MAIN BLOCK
*
* SKIP OVER THE UNUSED AND BOOTSTRAP AREAS
*
        LDA     #$F0            FORCE LS. ADDRESS BYTE TO $F0
        STA     ADDR+1          .
        STA     ADDRLO          MAKE SURE PORT A GETS UPDATED AS WELL
        BRN     *               DUMMY INSTRUCTION TO AVOID OPTION REG
*
        LDA     #$1F            FORCE MS. ADDRESS BYTE TO $1F
INMAIN  STA     ADDR            UPDATE MS. ADDRESS
*
* COMBINE MS. ADDRESS BYTE WITH OLD TSC & LED LEVELS & UPDATE PORT C
*
        LDA     PORTC           GET OLD MS. ADDRESS AND CONTROL LINES
        AND     #CMASK          MASK OUT ADDRESS LINES
        ORA     ADDR            COMBINE CONTROL LINES WITH NEW ADDRES
        STA     ADDRHI          UPDATE PORT C
GOBACK  RTS
*********************************************************************
        FCB     0               ONE SPARE BYTE!
*
        ORG     $1FEE
*
RESET   FDB     START           RESET VECTOR
*
        END

?
```

Memory Map Diagram Clarification

Reference Document: MC68HC705C8/D Rev. 1, page 2-7

Tracker Number: HC705C8.015 Revision: 1.00

The memory map diagram (Figure 2-3, page 2-7) shows only one address as the separator ($002F) between the unused 16 bytes and the 32 bytes of RAM. The question arises about whether the address $002F is the end of the unused section or the beginning of the 32 bytes of RAM.

The diagram should be changed to indicate that the address $002F is the end of the unused section and that the 32 bytes of RAM start at address $0030.

Bootloader for Mask Set 1C11C, 2C11C, 3C11C, 6C11C, 7C11C, and 9C11C

Reference Document: Not applicable

Tracker Number: HC705C8.021 Revision: 1.00

```
***********************************************************
*                                                         *
*             68HC705C8 EPROM BOOTLOADER PROGRAM          *
*             ==================================          *
*                                                         *
*              This version:-          - REV 2            *
*                                Fixed initial prog time  *
*                             11/17/87 - REV 1            *
*                                                         *
***********************************************************
*
* I/O DEFINITIONS
*
PORTA   EQU     $00     PORT A DATA
PORTB   EQU     $01     PORT B DATA
PORTC   EQU     $02     PORT C DATA
PORTD   EQU     $03     PORT D DATA   (Input Only!)
DDRA    EQU     $04     PORT A DDR
DDRB    EQU     $05     PORT B DDR
DDRC    EQU     $06     PORT C DDR
*
* SERIAL PERIPHERAL INTERFACE REGISTERS
*
SPCR    EQU     $0A     SERIAL PERIPHERAL CONTROL
SPSR    EQU     $0B     SERIAL PERIPHERAL STATUS
SPDAT   EQU     $0C     SERIAL PERIPHERAL DATA
*
* SERIAL COMMUNICATIONS INTERFACE REGISTERS
*
BAUD    EQU     $0D     BAUD RATE CONTROL
SCCR1   EQU     $0E     SERIAL COMM'S CONTROL REGISTER 1
SCCR2   EQU     $0F     SERIAL COMM'S CONTROL REGISTER 2
SCSR    EQU     $10     SERIAL COMM'S STATUS
SCDAT   EQU     $11     SERIAL COMM'S DATA
*
```

```
* TIMER REGISTERS
*
TIMCR   EQU     $12     TIMER CONTROL
TIMSR   EQU     $13     TIMER STATUS
IPCAPH  EQU     $14     INPUT CAPTURE  (High Byte)
IPCAPL  EQU     $15     INPUT CAPTURE  (Low  Byte)
OPCOMH  EQU     $16     OUTPUT COMPARE (High Byte)
OPCOML  EQU     $17     OUTPUT COMPARE (Low  Byte)
COUNTH  EQU     $18     COUNTER (High Byte)
COUNTL  EQU     $19     COUNTER (Low  Byte)
DUALTH  EQU     $1A     DUAL TM REGISTER (High Byte)
DUALTL  EQU     $1B     DUAL TM REGISTER (Low  Byte)
*
* EPROM CONTROL REGISTER
*
PROG    EQU     $1C     EPROM CONTROL
*
* MEMORY MAP DEFINITIONS
*
EPROM0  EQU     $20     BASE OF PAGE ZERO EPROM AREA
RAM     EQU     $50     BEGINNING OF RAM
STACK   EQU     $FF     STACK RESET ADDRESS
EPROM   EQU     $0100   BASE OF MAIN EPROM AREA
BOOTST  EQU     $1F00   START OF BOOTSTRAP ROM AREA
BOOTV   EQU     $1FE0   START OF BOOTSTRAP VECTOR AREA
VECTOR  EQU     $1FF0   START OF USER VECTOR AREA
*
* RAM VARIABLES
*
RAMSUB  EQU     RAM        LOCATION OF RAM SUBROUTINE
ADDR    EQU     RAMSUB+1   EXTENDED ADDRESS FOR RAM SUBROUTINE
DELAY   EQU     RAMSUB+4   INTER-BYTE DELAY
TIME    EQU     RAMSUB+5   PROGRAMMING PULSE WIDTH
*
* PORT A DEFINITIONS
*
ADDRLO  EQU     PORTA      LOW  ORDER ROM ADDRESSES A0 - A7
A5      EQU     5          BIT 5 :- ADDRESS LINE A5
*
* PORT B DEFINITIONS
*
DATAIN  EQU     PORTB      ROM DATA INPUT PORT
*
* PORT C DEFINITIONS
*
ADDRHI  EQU     PORTC      HIGH ORDER ROM ADDRESSES A8 - A12
VFYLED  EQU     5          BIT 5 DRIVES 'VERIFY' LED
PRGLED  EQU     6          BIT 6 DRIVES 'PROGRAMMING' LED
TSC     EQU     7          BIT 7 CONTROLS EXTERNAL ROM TRI-STATING..
*                          OUTPUT PIN - 0=MEMORY ENABLED  1= MEMORY
*                          TRI-STATED (FLOAT IF NOT USED).
*
* PORT D DEFINITIONS
*
MODES   EQU     PORTD      BOOTSTRAP MODE INPUT PORT
PIND2   EQU     2          BIT 2
PIND3   EQU     3          BIT 3
PIND4   EQU     4          BIT 4
PIND5   EQU     5          BIT 5
*
```

```
* MISCELLANEOUS DEFINITIONS
*
CMASK   EQU     %11100000  PORTC CONTROL LINES MASK
EPGM    EQU     0          PROG  BIT0; - Vpp CONTROL BIT
ERASED  EQU     $00        VALUE OF AN ERASED EPROM BYTE
INSTAT  EQU     %01100000  INITIAL PORT C LED STATUS
LAT     EQU     2          PROG  BIT2; - EPROM ADDRESS LATCH BIT
LATCH   EQU     %00000100  PROG  BIT2
MUL     EQU     $42        OP-CODE FOR MULTIPLY INSTRUCTION
OCF     EQU     6          TIMSR BIT6; - OUTPUT COMPARE FLAG
OLVL    EQU     0          TIMCR BIT0; - TIMER COMPARE OUTPUT LEVEL
RDRF    EQU     5          SCSR  BIT5; - RCV DATA REG FULL FLAG
TDRE    EQU     7          SCSR  BIT7; - XMIT DATA REG EMPTY FLAG
TEST    EQU     2          PORTD BIT2; - '0' GO BOOT,'1'GO $51 (RAM)
OPTION  EQU     $1FDF      OPTION REGISTER
TSTREG  EQU     $1F        TEST REGISTER
*
*********************************************************************
*
*    INITIAL REGISTER VALUES
*       FCB     %00100000   PORT A :- ROM ADDRESSES A7-A0 SET TO $20
*       FCB     $00         PORT B :- DON'T CARE
*       FCB     %01100000   PORT C :- ROM ENABLED/ LED'S OFF.....
*                               ...A12-A8 ZEROED
*       FCB     $00         PORT D :- DON'T CARE
*       FCB     %11111111   PORT A DDR :- ALL OUTPUTS
*       FCB     $00         PORT B DDR :- ALL INPUTS
*       FCB     %11111111   PORT C DDR :- ALL OUTPUTS
*       FCB     $00         SPCR   :- DISABLE OPERATION
*       FCB     $00         SPSR   :- DON'T CARE
*       FCB     $00         SPDAT  :- DON'T CARE
*       FCB     %00110000   BAUD   :- =4800Baud @2Mhz XTAL
*       FCB     %00000000   SCCR1  :- 8 DATA BITS
*       FCB     %00001100   SCCR2  :- INHIBIT INTERRUPTS, ENABLE...
*                                     RX/TX NO WAKE-UP, NO BREAK
*       FCB     $00         SCSR   :- DON'T CARE
*       FCB     $00         SCDAT  :- DON'T CARE
*       FCB     $00         TIMCR  :- DISABLE TIMER INTERRUPTS
*       FCB     $00         TIMSR  :- DON'T CARE
*       FCB     $00         IPCAPH :- DON'T CARE
*       FCB     $00         IPCAPL :- DON'T CARE
*       FCB     $00         OPCOMH :- DON'T CARE
*       FCB     $00         OPCOML :- DON'T CARE
*       FCB     $00         COUNTH :- DON'T CARE
*       FCB     $00         COUNTL :- DON'T CARE
*       FCB     $00         DUALTH :- DON'T CARE
*       FCB     $00         DUALTL :- DON'T CARE
*       FCB     $00         PROG   :- DISABLE PROGRAMMING
*
*
*   RAM AREA IS INITIALISED AS FOLLOWS;
*
*   LOCATION:-            INSTRUCTION:-
*
* RAMSUB  $50     $C7     STA    EPROM0
* ADDR    $51     $00
* ADDR+1  $52     $20
*         $53     $81     RTS
*
*   TIMER VARIABLES
*
* DELAY   $54       1          -
* TIME    $55       3          -
*
*
```

```
******************************************************************
        ORG     BOOTST
*
TABLE   FCB     $C7             'STA EXTENDED' INSTRUCTION
        FCB     $00             ADDRESS $0020
        FCB     $20             .
        FCB     $81             'RTS' INSTRUCTION
        FCB     1               1mS  INTER-BYTE DELAY CONSTANT
        FCB     $03             3 mS PROGRAMMING PULSE CONSTANT
*
START   EQU     *
*
* FIRST CHECK FOR SECURITY AND HANG IF ENABLED
*
        LDA     OPTION
        AND     #%00001000
        BEQ     NOSEC
        STOP
*
* THEN CHECK PORT D, BIT 2, TO SEE IF USER WISHES TO JUMP TO
* RAM OR JUMP INTO THE BOOTLOADER PROGRAM.
*
*
NOSEC   BRCLR   TEST,PORTD,BOOT
        JMP     RAM+1                GO TO RAM PROGRAM AT $0051
*
*
*
* SET UP PORTS, TIMER, RAM SUBROUTINE, AND RAM VARIABLES
*
*
BOOT    CLR     SCCR1           SET SCI TO 8 DATA BITS OPERATION
        LDX     #DDRA           POINT TO DDRA & INIT X FOR MULTIPLY
        COM     ,X              SET PORTA TO ALL OUTPUTS
        COM     DDRC            SET PORTC TO ALL OUTPUTS
*
        LDA     #%00001100      GET SCCR2 INITIAL VALUE
        STA     SCCR2           SET UP SCCR2
*
*       GET BAUD INITIAL VALUE (N.B. MUL NOT SUPPORTED BY ASSEMBLER!)
        FCB     MUL             X=0, A=%00110000 <== X=4, A=%00001100
        STA     BAUD            SET UP BAUD
*
        LSLA                    INITIAL PORT C VALUE (%01100000)
        STA     PORTC           ENABLE ROM, TURN OFF LED'S, ZERO..
*                               ZERO A12-A8
        LDA     #EPROM0         GET INITIAL PORT A (A7 - A0) VALUE
        STA     ,X              SET UP PORT A
*
*  INITIALISE RAM SUBROUTINE AND VARIABLES THEN JUST FILL RAM
*
MOVE    LDA     TABLE,X         GET A BYTE FROM THE TABLE
        STA     RAM,X           MOVE IT INTO RAM
        INCX                    POINT TO THE NEXT BYTE TO BE MOVED
        BNE     MOVE            KEEP MOVING UNTIL ALL ARE IN PLACE
*
*
*
        BRCLR   PIND5,MODES,MOVED
        BRSET   PIND3,MODES,SEC
*
*DO THE GATE STRESS TEST
*
```

```
GTEST   LDA     #$15    MOE/TS1=0,STE/ST0=1 ALSO RCKH=1
        STA     TSTREG  ENABLE GATE STRESS TEST
*                       ...AND OUTPUT XMIT CLOCK ON TDO
        BSET    0,ADDR  SET UP FOR ADDRESS $0120
        BRCLR   PIND4,MODES,NOCOM
        COM     ADDR+1  USE ALTERNATE ADDRESS ($01D0)
NOCOM   BSR     ZAP
        WAIT
*
*PROGRAM THE SECURITY BIT
*
SEC     COMA            A HAS $06 FROM RAM MOVE
        BSET    LAT,PROG
        STA     OPTION
        BSR     ZAPSUB
*
*********************************************************************
*
* THE BOOTLOADER PROGRAM HAS 5 MODES OF OPERATION:
*
*   I. PROGRAM/VERIFY - PERFORMS A NORMAL PROGRAM CYCLE FOLLOWED BY A
*                       VERIFY CYCLE WHICH HANGS IF THE EPROM IS NOT
*                       CORRECTLY PROGRAMMED. EITHER 1ms OR 5ms PULSE
*                       WIDTH CAN BE SELECTED.
*
*  II. VERIFY - PERFORMS ONLY A VERIFY CYCLE WHICH HANGS IF THE EPROM
*               IS NOT CORRECTLY PROGRAMMED.
*
* III. LOAD RAM - LOADS A PROGRAM FROM SCI INTO THE RAM
*                 THEN JUMPS TO RAM TO EXECUTE THE PROGRAM.
*
*  IV. DUMP EPROM - DUMPS THE EPROM CONTENTS OF THE 705C8 TO THE SCI
*
*   V. SECURE THE PART - PROGRAMS ONLY THE SECURITY BIT AND THEN
*                        EITHER DOES ANOTHER VERIFY (NOT THE SEC BIT)
*                        OR HANGS IN THE DUMP EPROM MODE
*
* WHEN COMING OUT OF RESET INTO THE BOOTLOADER PROGRAM (ASSUMING THAT
* PORT D PIN 2 ALLOWS YOU TO ENTER THE BOOTLOADER) THE STATE OF
* PORT D PINS 3, 4 AND 5 DETERMINES WHICH MODE OF OPERATION THE
* PROGRAM WILL ENTER.
*
* THE GATE STRESS TEST CAN ALSO BE INVOKED THROUGH THE BOOTLOADER.
*
*    |--------------------------|---------------------|
*    |         PORT D           |                     |
*    |-------|--------|---------|        MODE         |
*    | PIN 5 | PIN 4  |  PIN 3  |                     |
*    |-------|--------|---------|--------------------------------|
*    |   0   |   0    |    0    |   PROGRAM/VERIFY  15ms PULSE  |
*    |-------|--------|---------|--------------------------------|
*    |   0   |   0    |    1    |   VERIFY                       |
*    |-------|--------|---------|--------------------------------|
*    |   0   |   1    |    0    |   LOAD RAM                     |
*    |-------|--------|---------|--------------------------------|
*    |   0   |   1    |    1    |   DUMP EPROM                   |
*    |=======|========|=========|================================|
*    |   1   |   0    |    0    |   GATE STRESS TEST             |
*    |-------|--------|---------|--------------------------------|
*    |   1   |   0    |    1    |   SECURE/VERIFY                |
*    |-------|--------|---------|--------------------------------|
*    |   1   |   1    |    0    |   GATE STRESS TEST             |
*    |-------|--------|---------|--------------------------------|
*    |   1   |   1    |    1    |   SECURE/DUMP EPROM            |
*    |=======|========|=========|================================|
*
*********************************************************************
```

```
*
*
* CHECK PORT D PINS 3, 4 AND 5 TO DETERMINE WHICH MODE TO ENTER
*
*
MOVED   BRCLR   PIND4,MODES,PRGVER      DO A PROGRAM/VERIFY OR VERIFY
        BRCLR   PIND3,MODES,LDRAM       DO A LOAD RAM - EXECUTE CYCLE
*
* DUMP THE CONTENTS OF THE 705C8 EPROM TO THE SCI OUTPUT.
*        ( ASSUMES 'RAMSUB' CONTAINS $C7 )
*
DMPEPR  DEC     RAMSUB          CHANGE 'STA' TO 'LDA' EXTENDED ($C6).
DUMPIT  JSR     RAMSUB          READ ONE BYTE OF EPROM
WAITTX  BRCLR   TDRE,SCSR,WAITTX  WAIT FOR TRANSMIT REGISTER TO EMPTY
        STA     SCDAT           SEND EPROM DATA TO SERIAL OUTPUT
        BSR     NXTADR          MOVE TO NEXT ADDRESS
        BNE     DUMPIT          IF NOT FINISHED, CONTINUE
        BRA     *               HANG WHEN FINISHED.
*
*
* CHOOSE BETWEEN PROGRAM/VERIFY AND JUST VERIFY MODES
*
*
PRGVER  BRSET   PIND3,MODES,VERIFY      DO A VERIFY CYCLE
        BCLR    PRGLED,PORTC            LIGHT 'PROGRAMMING' LED
*
*
* PROGRAM THE EPROM WITH THE CONTENTS OF THE EXTERNAL ROM
*
*
PRGLOP  BSR     PRGSUB                  PROGRAM ONE EPROM BYTE
        BSR     NXTADR                  POINT TO NEXT ADDRESS
        BNE     PRGLOP                  KEEP PROGRAMMING UNTIL DONE
        STA     ADDR+1                  RESET LOW ORDER  ADDR TO $20
        CLR     ADDR                    RESET HIGH ORDER ADDR TO $00
*
*
* VERIFY THE EPROM CONTENTS AGAINST EXTERNAL MEMORY.
*        ( ASSUMES 'RAMSUB' CONTAINS $C7 )
*
VERIFY  LDA     #INSTAT         GET INITIAL LED STATUS FOR A VERIFY
COMPAR  INC     RAMSUB          CHANGE 'STA' TO 'EOR' EXTENDED ($C8).
        STA     PORTC           PLACE LED AND ROM STATUS ON PORT PINS
        BSET    A5,ADDRLO       SET A7 - A0 TO $20 (START OF EPROM)
*
CHECK   LDA     DATAIN          READ A BYTE FROM THE EXTERNAL MEMORY
        JSR     RAMSUB          COMPARE TO AN EPROM BYTE
        BNE     *               HANG IF THEY DON'T MATCH
        BSR     NXTADR          POINT TO NEXT ADDRESS TO BE COMPARED
        BNE     CHECK           KEEP CHECKING BYTES UNTIL EPROM END
*
DONE    BCLR    VFYLED,PORTC    INDICATE EPROM VERIFIED AS CORRECT
        BRA     *
*
*
*
* LOAD THE RAM WITH A USER'S PROGRAM VIA THE SCI.
*
*    THE DATA SHOULD BE IN THE FORM OF 1 START-BIT, 8 DATA-BITS,
* 1 STOP-BIT. THE FIRST BYTE SHOULD BE A COUNT OF THE TOTAL NUMBER
* OF BYTES TO BE SENT, INCLUDING THAT BYTE. THE FIRST BYTE IS
* LOADED INTO $50 SO THE FIRST PROGRAM BYTE WILL BE LOADED INTO
* $51. THAT IS WHERE PROGRAM EXECUTION WILL BEGIN.
*    IF A COUNT IS USED THAT IS GREATER THAN ( PROGRAM LENGTH + 1 )
* THEN THE ROUTINE WILL HANG AFTER THE LAST PROGRAM BYTE IS SENT.
* THIS CAN BE USED TO HOLD OFF EXECUTION OF THE PROGRAM UNTIL BIT-2
* OF PORTD IS SET AND RESET IS ASSERTED.
```

```
*
*
LDRAM   LDX     #RAM    POINT TO START OF RAM
WAITRX  BRCLR   RDRF,SCSR,WAITRX   WAIT FOR RX REGISTER TO FILL
        LDA     SCDAT   READ DATA BYTE FROM RX REGISTER
        STA     ,X      STORE THE DATA IN RAM
        INCX            MOVE TO NEXT RAM LOCATION
        DEC     RAM     DECREMENT THE PROGRAM SIZE COUNTER (1st BYTE)
        BNE     WAITRX  IF ENTIRE PROGRAM NOT LOADED, CONTINUE
JMPRAM  JMP     RAM+1   JUMP TO THE PROGRAM IN THE RAM
*
*******************************************************************
*                                                                 *
*                      S U B R O U T I N E S                      *
*                                                                 *
*******************************************************************
*
* PROGRAM AN EPROM ADDRESS WITH DATA RECEIVED FROM PORTB.
* THE ADDRESS TO BE PROGRAMMED SHOULD BE PLACED IN LOCATION
* 'ADDR' & 'ADDR+1'.
*
*
PRGSUB  LDA     DATAIN          READ DATA FROM PORTB
        BEQ     SKIP            RETURN IF EQUAL TO ERASED STATE ($00)
*
ZAPSUB  BSR     ZAP
        BCLR    EPGM,PROG      REMOVE Vpp FROM CIRCUIT
*
        LDA     DELAY           GET INTER-BYTE DELAY LENGTH
        BSR     DELNMS          DELAY BEFORE CONTINUING
        BCLR    LAT,PROG        CLEAR THE LAT BIT
SKIP    RTS
*
ZAP     BSET    LAT,PROG
        JSR     RAMSUB          WRITE ONE BYTE OF DATA
        BSET    EPGM,PROG      APPLY Vpp TO CIRCUIT
        LDA     TIME            GET PROGRAMMING PULSE LENGTH
*****************************************************************
*
*
* DELAY N mS SUBROUTINE. ON ENTRY, ACCUMULATOR SHOULD CONTAIN THE
* TIME DELAY WANTED IN MILLISECONDS. AT THE END OF THE DELAY BIT
* 'OLVL' WILL BE CLOCKED TO THE 'TCMP' PIN.
* ( ASSUMES 2MHz OSCILLATOR FREQUENCY ).
*
*
DELNMS  ADD     DUALTH          ADD WANTED DELAY TO CURRENT COUNT
        STA     OPCOMH          STORE NEW VALUE INTO MS.COMPARE REG
        LDA     TIMSR           ACCESS TIMER STATUS REGISTER.......
        LDA     #$00
        STA     OPCOML          ...THEN WRITE TO LS.COMPARE REGISTER
HOLD    BRCLR   OCF,TIMSR,HOLD  NOW WAIT FOR COMPARE FLAG TO BE SET
        RTS                     RETURN AFTER WANTED DELAY
*
*******************************************************************
*
*  NXTADR SUBROUTINE
*
*  COMPUTES NEXT EPROM ADDRESS TO BE PROGRAMMED, VERIFIED, OR DUMPED.
*  PUTS THIS ADDRESS ON PORTS A (LS BYTE) AND C (MS BYTE) AND
*  UPDATES RAMSUB ALSO. SKIPS THE RAM, BOOTSTRAP AND UNUSED AREAS.
*  RETURNS WITH Z=1 IF THE COMPUTED ADDRESS IS = $2000, MEANING THAT
*  A PASS THROUGH THE MEMORY MAP HAS BEEN COMPLETED. OTHERWISE Z=0.
*
```

```
NXTADR  LDA     ADDR            GET MS.BYTE OF LAST ADDRESS USED
        BNE     NOT04F          BRANCH IF NOT IN PAGE ZERO EPROM AREA
        LDX     ADDR+1          GET LS.BYTE OF LAST ADDRESS USED
        CPX     #RAM-1          LAST BYTE OF PAGE ZERO EPROM?
        BNE     NOT04F          BRANCH IF NOT $004F
*
* MUST HAVE JUST ACCESSED LAST PAGE ZERO EPROM LOCATION IF HERE SO
* FORCE NEXT ADDRESS INCREMENT TO POINT TO MAIN EPROM AREA AT $0100.
*
        LDX     #$0100-1        POINT TO ADDRESS BELOW THE ONE WANTED
        STX     ADDR+1          PLACE IN LOCATION TO BE INCREMENTED
        STX     ADDRLO          PRESET PORTA PRIOR TO INCREMENT
*
* 16 BIT ADDRESS INCREMENT
*
NOT04F  INC     ADDR+1          INCREMENT LS. ADDRESS BYTE
        INC     ADDRLO          UPDATE LS.ADDRESS AT PORT A
        BNE     GOBACK          RETURN IF A PAGE BOUNDARY NOT REACHED
        INCA                    INCREMENT MS. ADDRESS
*
* LOOK OUT FOR HAVING GONE THROUGH THE ENTIRE MEMORY MAP.
*
        CMP     #$20            WAS THAT THE END OF MEMORY ( $1FFF )?
        BEQ     GOBACK          EXIT WITH Z=1 IF THE END WAS REACHED.
*
* LOOK OUT FOR HAVING ACCESSED THE LAST LOCATION IN THE MAIN BLOCK
*
        CMP     #$1F            WAS THAT THE END OF THE MAIN BLOCK
        BNE     INMAIN          BRANCH IF STILL WITHIN THE MAIN BLOCK
*
* SKIP OVER THE UNUSED AND BOOTSTRAP AREAS
*
        LDA     #$F0            FORCE LS. ADDRESS BYTE TO $F0
        STA     ADDR+1          .
        STA     ADDRLO          MAKE SURE PORT A GETS UPDATED AS WELL
        BRN     *               DUMMY INSTRUCTION TO AVOID OPTION REG
*
        LDA     #$1F            FORCE MS. ADDRESS BYTE TO $1F
INMAIN  STA     ADDR            UPDATE MS. ADDRESS
*
* COMBINE MS. ADDRESS BYTE WITH OLD TSC & LED LEVELS & UPDATE PORT C
*
        LDA     PORTC           GET OLD MS. ADDRESS AND CONTROL LINES
        AND     #CMASK          MASK OUT ADDRESS LINES
        ORA     ADDR            COMBINE CONTROL LINES WITH NEW ADDRES
        STA     ADDRHI          UPDATE PORT C
GOBACK  RTS
*********************************************************************
        FCB     0               ONE SPARE BYTE!
*
        ORG     $1FEE
*
RESET   FDB     START           RESET VECTOR
*
        END
?
```

MC68HC705C8A

Revision History

Date	Revision	Description
5/4/95	1.00	Includes trackers HC705C8A.003, 68HC705C8AMSE1, 68HC705C8AMSE2 (R1).

Bootloader for Mask Set 0E20T, 1E20T, 2E20T, 3E20T, 0E79R, 1E79R, 2E79R, and 3E79R

Reference Document: Not applicable

Tracker Number: HC705C8A.003 **Revision: 1.00**

```
***********************************************************
*                                                         *
*             68HC705C8A EPROM BOOTLOADER PROGRAM         *
*             ==================================          *
*                                                         *
*  REV 1 -  05/18/92   NCN                                *
*           STARTED FROM 705BOOT.5                        *
*           CHANGED NXTADR TO ALLOW USE OF IRQ VECTOR FOR *
*           KEYSCAN TESTING IN "JMP RAM MODE".            *
*  REV 2 - 12/11/92    NCN                                *
*           FIXED PROGRAMMING OF THE SECURITY BIT.        *
*           BY CHANGING BSR  ZAPSUB AT ADDRESS 1FB7       *
*           TO  BSR  ZAPSEC. 1 BYTE CHANGED AT ADDRESS    *
*           $1FB8 FROM $E5 TO $EE.                        *
*                                                         *
*  REV 3 -  4/19/93    NCN                                *
*           FIXED GATE STRESS BY CHANGING                 *
*           "LDA #$20" AT ADDRESS $1F36 TO "LDA #$40".    *
***********************************************************
*
* I/O DEFINITIONS
*
PORTA   EQU     $00     PORT A DATA
PORTB   EQU     $01     PORT B DATA
PORTC   EQU     $02     PORT C DATA
PORTD   EQU     $03     PORT D DATA   (Input Only!)
DDRA    EQU     $04     PORT A DDR
DDRB    EQU     $05     PORT B DDR
DDRC    EQU     $06     PORT C DDR
*
* SERIAL PERIPHERAL INTERFACE REGISTERS
*
SPCR    EQU     $0A     SERIAL PERIPHERAL CONTROL
SPSR    EQU     $0B     SERIAL PERIPHERAL STATUS
SPDAT   EQU     $0C     SERIAL PERIPHERAL DATA
*
```

```
* SERIAL COMMUNICATIONS INTERFACE REGISTERS
*
BAUD    EQU     $0D     BAUD RATE CONTROL
SCCR1   EQU     $0E     SERIAL COMM'S CONTROL REGISTER 1
SCCR2   EQU     $0F     SERIAL COMM'S CONTROL REGISTER 2
SCSR    EQU     $10     SERIAL COMM'S STATUS
SCDAT   EQU     $11     SERIAL COMM'S DATA
*
* TIMER REGISTERS
*
TIMCR   EQU     $12     TIMER CONTROL
TIMSR   EQU     $13     TIMER STATUS
IPCAPH  EQU     $14     INPUT CAPTURE  (High Byte)
IPCAPL  EQU     $15     INPUT CAPTURE  (Low  Byte)
OPCOMH  EQU     $16     OUTPUT COMPARE (High Byte)
OPCOML  EQU     $17     OUTPUT COMPARE (Low  Byte)
COUNTH  EQU     $18     COUNTER (High Byte)
COUNTL  EQU     $19     COUNTER (Low  Byte)
DUALTH  EQU     $1A     DUAL TM REGISTER (High Byte)
DUALTL  EQU     $1B     DUAL TM REGISTER (Low  Byte)
*
* EPROM CONTROL REGISTER
*
PROG    EQU     $1C     EPROM CONTROL
*
* MEMORY MAP DEFINITIONS
*
EPROM0  EQU     $20     BASE OF PAGE ZERO EPROM AREA
RAM     EQU     $50     BEGINNING OF RAM
STACK   EQU     $FF     STACK RESET ADDRESS
EPROM   EQU     $0100   BASE OF MAIN EPROM AREA
BOOTST  EQU     $1F00   START OF BOOTSTRAP ROM AREA
BOOTV   EQU     $1FE0   START OF BOOTSTRAP VECTOR AREA
VECTOR  EQU     $1FF0   START OF USER VECTOR AREA
*
* RAM VARIABLES
*
RAMSUB  EQU     RAM        LOCATION OF RAM SUBROUTINE
ADDR    EQU     RAMSUB+1   EXTENDED ADDRESS FOR RAM SUBROUTINE
LOOP    EQU     RAMSUB+4   INTER-BYTE DELAY
TIME    EQU     RAMSUB+5   PROGRAMMING PULSE WIDTH
*
* PORT A DEFINITIONS
*
ADDRLO  EQU     PORTA      LOW  ORDER ROM ADDRESSES A0 - A7
A5      EQU     5          BIT 5 :- ADDRESS LINE A5
*
* PORT B DEFINITIONS
*
DATAIN  EQU     PORTB      ROM DATA INPUT PORT
*
* PORT C DEFINITIONS
*
ADDRHI  EQU     PORTC      HIGH ORDER ROM ADDRESSES A8 - A12
VFYLED  EQU     5          BIT 5 DRIVES 'VERIFY' LED
PRGLED  EQU     6          BIT 6 DRIVES 'PROGRAMMING' LED
TSC     EQU     7          BIT 7 CONTROLS EXTERNAL ROM TRI-STATING..
*                          OUTPUT PIN - 0=MEMORY ENABLED  1= MEMORY
*                          TRI-STATED (FLOAT IF NOT USED).
*
* PORT D DEFINITIONS
*
MODES   EQU     PORTD      BOOTSTRAP MODE INPUT PORT
PIND2   EQU     2          BIT 2
PIND3   EQU     3          BIT 3
PIND4   EQU     4          BIT 4
PIND5   EQU     5          BIT 5
```

```
*
* MISCELLANEOUS DEFINITIONS
*
CMASK   EQU     %11100000  PORTC CONTROL LINES MASK
EPGM    EQU     0          PROG  BIT0; - Vpp CONTROL BIT
ERASED  EQU     $00        VALUE OF AN ERASED EPROM BYTE
INSTAT  EQU     %01100000  INITIAL PORT C LED STATUS
LAT     EQU     2          PROG  BIT2; - EPROM ADDRESS LATCH BIT
LATCH   EQU     %00000100  PROG  BIT2
MUL     EQU     $42        OP-CODE FOR MULTIPLY INSTRUCTION
OCF     EQU     6          TIMSR BIT6; - OUTPUT COMPARE FLAG
OLVL    EQU     0          TIMCR BIT0; - TIMER COMPARE OUTPUT LEVEL
RDRF    EQU     5          SCSR  BIT5; - RCV DATA REG FULL FLAG
TDRE    EQU     7          SCSR  BIT7; - XMIT DATA REG EMPTY FLAG
TEST    EQU     2          PORTD BIT2; - '0' GO BOOT,'1'GO $51 (RAM)
OPTION  EQU     $1FDF      OPTION REGISTER
TSTREG  EQU     $1F        TEST REGISTER
*
*********************************************************************
*
*    INITIAL REGISTER VALUES
*       FCB     %00100000   PORT A :- ROM ADDRESSES A7-A0 SET TO $20
*       FCB     $00         PORT B :- DON'T CARE
*       FCB     %01100000   PORT C :- ROM ENABLED/ LED'S OFF.....
*                               ...A12-A8 ZEROED
*       FCB     $00         PORT D :- DON'T CARE
*       FCB     %11111111   PORT A DDR :- ALL OUTPUTS
*       FCB     $00         PORT B DDR :- ALL INPUTS
*       FCB     %11111111   PORT C DDR :- ALL OUTPUTS
*       FCB     $00         SPCR   :- DISABLE OPERATION
*       FCB     $00         SPSR   :- DON'T CARE
*       FCB     $00         SPDAT  :- DON'T CARE
*       FCB     %00110000   BAUD   :- =4800Baud @2Mhz XTAL
*       FCB     %00000000   SCCR1  :- 8 DATA BITS
*       FCB     %00001100   SCCR2  :- INHIBIT INTERRUPTS, ENABLE...
*                                     RX/TX NO WAKE-UP, NO BREAK
*       FCB     $00         SCSR   :- DON'T CARE
*       FCB     $00         SCDAT  :- DON'T CARE
*       FCB     $00         TIMCR  :- DISABLE TIMER INTERRUPTS
*       FCB     $00         TIMSR  :- DON'T CARE
*       FCB     $00         IPCAPH :- DON'T CARE
*       FCB     $00         IPCAPL :- DON'T CARE
*       FCB     $00         OPCOMH :- DON'T CARE
*       FCB     $00         OPCOML :- DON'T CARE
*       FCB     $00         COUNTH :- DON'T CARE
*       FCB     $00         COUNTL :- DON'T CARE
*       FCB     $00         DUALTH :- DON'T CARE
*       FCB     $00         DUALTL :- DON'T CARE
*       FCB     $00         PROG   :- DISABLE PROGRAMMING
*
*
*   RAM AREA IS INITIALISED AS FOLLOWS;
*
*   LOCATION:-            INSTRUCTION:-
*
* RAMSUB  $50     $C7     STA    EPROM0
* ADDR    $51     $00
* ADDR+1  $52     $20
*         $53     $81     RTS
*
*   TIMER VARIABLES
*
* LOOP    $54       2          -
* TIME    $55       1          -
*
*
```

```
******************************************************************
         ORG     BOOTST
*
TABLE    FCB     $C7             'STA EXTENDED' INSTRUCTION
         FCB     $00             ADDRESS $0020
         FCB     $20             .
         FCB     $81             'RTS' INSTRUCTION
         FCB     2               2 PASSES THRU PROG
         FCB     $01             1 mS PROGRAMMING PULSE CONSTANT
*
START    EQU     *
*
* FIRST CHECK FOR SECURITY AND HANG IF ENABLED
*
         LDA     OPTION
         AND     #%00001000
         BEQ     NOSEC
         BRA    *
*
* THEN CHECK PORT D, BIT 2, TO SEE IF USER WISHES TO JUMP TO
* RAM OR JUMP INTO THE BOOTLOADER PROGRAM.
*
*
NOSEC    BRCLR   TEST,PORTD,BOOT
         JMP     RAM+1                GO TO RAM PROGRAM AT $0051
*
*
*
* SET UP PORTS, TIMER, RAM SUBROUTINE, AND RAM VARIABLES
*
*
BOOT     CLR     SCCR1           SET SCI TO 8 DATA BITS OPERATION
         LDX     #DDRA           POINT TO DDRA & INIT X FOR MULTIPLY
         COM     ,X              SET PORTA TO ALL OUTPUTS
         COM     DDRC            SET PORTC TO ALL OUTPUTS
*
         LDA     #%00001100      GET SCCR2 INITIAL VALUE
         STA     SCCR2           SET UP SCCR2
*
*       GET BAUD INITIAL VALUE (N.B. MUL NOT SUPPORTED BY ASSEMBLER!)
         FCB     MUL             X=4, A=%00001100 ==> X=0, A=%00110000
         STA     BAUD            SET UP BAUD
*
         LSLA                    INITIAL PORT C VALUE (%01100000)
         STA     PORTC           ENABLE ROM, TURN OFF LED'S, ZERO..
*                               ZERO A12-A8
         LDA     #EPROM0         GET INITIAL PORT A (A7 - A0) VALUE
         STA     ,X              SET UP PORT A
*
*  INITIALISE RAM SUBROUTINE AND VARIABLES THEN JUST FILL RAM
*
MOVE     LDA     TABLE,X         GET A BYTE FROM THE TABLE
         STA     RAM,X           MOVE IT INTO RAM
         INCX                    POINT TO THE NEXT BYTE TO BE MOVED
         BNE     MOVE            KEEP MOVING UNTIL ALL ARE IN PLACE
*
*
*
         BRCLR   PIND5,MODES,MOVED
         BRSET   PIND3,MODES,SEC
*
```

```
*DO THE GATE STRESS TEST
*
GTEST     LDA     #$40    EPMTST = 1, TS1:TS0 = 00
          STA     PROG    ENABLE GATE STRESS TEST
          BSET    0,ADDR  SET UP FOR ADDRESS $0120
          BRCLR   PIND4,MODES,NOCOM
          COM     ADDR+1  USE ALTERNATE ADDRESS ($01DF)
NOCOM     BSR     ZAP
          WAIT
*
*PROGRAM THE SECURITY BIT
*
SEC       LDA    #$08            SET SECURITY BIT
          BSET    LAT,PROG
          BRA    CONSEC
          NOP
*
*********************************************************************
*
* THE BOOTLOADER PROGRAM HAS 5 MODES OF OPERATION:
*
*   I. PROGRAM/VERIFY - PERFORMS A NORMAL PROGRAM CYCLE FOLLOWED BY A
*                       VERIFY CYCLE WHICH HANGS IF THE EPROM IS NOT
*                       CORRECTLY PROGRAMMED. EITHER 1ms OR 5ms PULSE
*                       WIDTH CAN BE SELECTED.
*
*  II. VERIFY - PERFORMS ONLY A VERIFY CYCLE WHICH HANGS IF THE EPROM
*               IS NOT CORRECTLY PROGRAMMED.
*
* III. LOAD RAM - LOADS A PROGRAM FROM SCI INTO THE RAM
*                 THEN JUMPS TO RAM TO EXECUTE THE PROGRAM.
*
*  IV. DUMP EPROM - DUMPS THE EPROM CONTENTS OF THE 705C8 TO THE SCI
*
*   V. SECURE THE PART - PROGRAMS ONLY THE SECURITY BIT AND THEN
*                        EITHER DOES ANOTHER VERIFY (NOT THE SEC BIT)
*                        OR HANGS IN THE DUMP EPROM MODE
*
* WHEN COMING OUT OF RESET INTO THE BOOTLOADER PROGRAM (ASSUMING THAT
* PORT D PIN 2 ALLOWS YOU TO ENTER THE BOOTLOADER) THE STATE OF
* PORT D PINS 3, 4 AND 5 DETERMINES WHICH MODE OF OPERATION THE
* PROGRAM WILL ENTER.
*
* THE GATE STRESS TEST CAN ALSO BE INVOKED THROUGH THE BOOTLOADER.
*
*    |--------------------------|---------------------|
*    |         PORT D           |                     |
*    |-------|--------|---------|       MODE          |
*    | PIN 5 | PIN 4  |  PIN 3  |                     |
*    |-------|--------|---------|------------------------------|
*    |   0   |   0    |    0    |   PROGRAM/VERIFY  15ms PULSE  |
*    |-------|--------|---------|------------------------------|
*    |   0   |   0    |    1    |   VERIFY                      |
*    |-------|--------|---------|------------------------------|
*    |   0   |   1    |    0    |   LOAD RAM                    |
*    |-------|--------|---------|------------------------------|
*    |   0   |   1    |    1    |   DUMP EPROM                  |
*    |=======|========|=========|==============================|
*    |   1   |   0    |    0    |   GATE STRESS TEST            |
*    |-------|--------|---------|------------------------------|
*    |   1   |   0    |    1    |   SECURE/VERIFY               |
*    |-------|--------|---------|------------------------------|
*    |   1   |   1    |    0    |   GATE STRESS TEST            |
*    |-------|--------|---------|------------------------------|
*    |   1   |   1    |    1    |   SECURE/DUMP EPROM           |
*    |=======|========|=========|==============================|
*
```

```
*********************************************************************
*
*
* CHECK PORT D PINS 3, 4 AND 5 TO DETERMINE WHICH MODE TO ENTER
*
*
MOVED     BRCLR   PIND4,MODES,PRGVER      DO A PROGRAM/VERIFY OR VERIFY
          BRCLR   PIND3,MODES,LDRAM       DO A LOAD RAM - EXECUTE CYCLE
*
* DUMP THE CONTENTS OF THE 705C8 EPROM TO THE SCI OUTPUT.
*        ( ASSUMES 'RAMSUB' CONTAINS $C7 )
*
DMPEPR    DEC     RAMSUB          CHANGE 'STA' TO 'LDA' EXTENDED ($C6).
DUMPIT    JSR     RAMSUB          READ ONE BYTE OF EPROM
WAITTX    BRCLR   TDRE,SCSR,WAITTX  WAIT FOR TRANSMIT REGISTER TO EMPTY
          STA     SCDAT           SEND EPROM DATA TO SERIAL OUTPUT
          BSR     NXTADR          MOVE TO NEXT ADDRESS
          BNE     DUMPIT          IF NOT FINISHED, CONTINUE
          WAIT                    HANG WHEN FINISHED.
*
*
* CHOOSE BETWEEN PROGRAM/VERIFY AND JUST VERIFY MODES
*
*
PRGVER    BRSET   PIND3,MODES,VERIFY      DO A VERIFY CYCLE
          BCLR    PRGLED,PORTC            LIGHT 'PROGRAMMING' LED
*
*
* PROGRAM THE EPROM WITH THE CONTENTS OF THE EXTERNAL ROM
*
*
PRGLOP    BSR     PRGSUB                  PROGRAM ONE EPROM BYTE
          BSR     NXTADR                  POINT TO NEXT ADDRESS
          BNE     PRGLOP                  KEEP PROGRAMMING UNTIL DONE
          STA     ADDR+1                  RESET LOW ORDER  ADDR TO $20
          STA    PORTC                RESET HIGH ORDER ADDRESS ON PORTC TO $20
          CLR     ADDR                    RESET HIGH ORDER ADDR TO $00
          BSET   A5,ADDRLO            PUT $20 ON PORT A
          DEC    LOOP                 DECREMENT LOOP COUNTER
          BNE    PRGLOP               RELOAD PARAM. FOR 2ND PASS
*
*
* VERIFY THE EPROM CONTENTS AGAINST EXTERNAL MEMORY.
*        ( ASSUMES 'RAMSUB' CONTAINS $C7 )
*
VERIFY    LDA     #INSTAT         GET INITIAL LED STATUS FOR A VERIFY
COMPAR    INC     RAMSUB          CHANGE 'STA' TO 'EOR' EXTENDED ($C8).
          STA     PORTC           PLACE LED AND ROM STATUS ON PORT PINS
          BSET    A5,ADDRLO       SET A7 - A0 TO $20 (START OF EPROM)
*
CHECK     LDA     DATAIN          READ A BYTE FROM THE EXTERNAL MEMORY
          JSR     RAMSUB          COMPARE TO AN EPROM BYTE
          BNE     *               HANG IF THEY DON'T MATCH
          BSR     NXTADR          POINT TO NEXT ADDRESS TO BE COMPARED
          BNE     CHECK           KEEP CHECKING BYTES UNTIL EPROM END
*
DONE      BCLR    VFYLED,PORTC    INDICATE EPROM VERIFIED AS CORRECT
          STOP
*
*
*
```

```
* LOAD THE RAM WITH A USER'S PROGRAM VIA THE SCI.
*
*    THE DATA SHOULD BE IN THE FORM OF 1 START-BIT, 8 DATA-BITS,
* 1 STOP-BIT. THE FIRST BYTE SHOULD BE A COUNT OF THE TOTAL NUMBER
* OF BYTES TO BE SENT, INCLUDING THAT BYTE. THE FIRST BYTE IS
* LOADED INTO $50 SO THE FIRST PROGRAM BYTE WILL BE LOADED INTO
* $51. THAT IS WHERE PROGRAM EXECUTION WILL BEGIN.
*    IF A COUNT IS USED THAT IS GREATER THAN ( PROGRAM LENGTH + 1 )
* THEN THE ROUTINE WILL HANG AFTER THE LAST PROGRAM BYTE IS SENT.
* THIS CAN BE USED TO HOLD OFF EXECUTION OF THE PROGRAM UNTIL BIT-2
* OF PORTD IS SET AND RESET IS ASSERTED.
*
*
LDRAM     LDX     #RAM    POINT TO START OF RAM
WAITRX    BRCLR   RDRF,SCSR,WAITRX   WAIT FOR RX REGISTER TO FILL
          LDA     SCDAT   READ DATA BYTE FROM RX REGISTER
          STA     ,X      STORE THE DATA IN RAM
          INCX            MOVE TO NEXT RAM LOCATION
          DEC     RAM     DECREMENT THE PROGRAM SIZE COUNTER (1st BYTE)
          BNE     WAITRX  IF ENTIRE PROGRAM NOT LOADED, CONTINUE
JMPRAM    JMP     RAM+1   JUMP TO THE PROGRAM IN THE RAM
*
********************************************************************
*                                                                  *
*                      S U B R O U T I N E S                       *
*                                                                  *
********************************************************************
*
* PROGRAM AN EPROM ADDRESS WITH DATA RECEIVED FROM PORTB.
* THE ADDRESS TO BE PROGRAMMED SHOULD BE PLACED IN LOCATION
* 'ADDR' & 'ADDR+1'.
*
*
PRGSUB    LDA     DATAIN          READ DATA FROM PORTB
          BEQ     SKIP            RETURN IF EQUAL TO ERASED STATE ($00)
*
ZAPSUB    BSR     ZAP
          CLR     PROG           REMOVE Vpp FROM CIRCUIT
*                                     CLEAR THE LAT BIT
SKIP      RTS
*
ZAP       BSET    LAT,PROG
          JSR     RAMSUB          WRITE ONE BYTE OF DATA
ZAPSEC    BSET    EPGM,PROG      APPLY Vpp TO CIRCUIT
          LDA     TIME            GET PROGRAMMING PULSE LENGTH
******************************************************************
*
*
* DELAY N mS SUBROUTINE. ON ENTRY, ACCUMULATOR SHOULD CONTAIN THE
* TIME DELAY WANTED IN MILLISECONDS. AT THE END OF THE DELAY BIT
* 'OLVL' WILL BE CLOCKED TO THE 'TCMP' PIN.
* ( ASSUMES 2MHz OSCILLATOR FREQUENCY ).
*
*
DELNMS    LDX    #$A6                 1 MS INNER LOOP
MS1       DECX
          BNE    MS1
          DECA                        DECREMENT OUTER LOOP
          BNE    DELNMS
          RTS                    RETURN AFTER WANTED DELAY
*
CONSEC    STA     OPTION
          BSR     ZAPSEC
*
```

```
********************************************************************
*
*  NXTADR SUBROUTINE
*
*  COMPUTES NEXT EPROM ADDRESS TO BE PROGRAMMED, VERIFIED, OR DUMPED.
*  PUTS THIS ADDRESS ON PORTS A (LS BYTE) AND C (MS BYTE) AND
*  UPDATES RAMSUB ALSO. SKIPS THE RAM, BOOTSTRAP AND UNUSED AREAS.
*  RETURNS WITH Z=1 IF THE COMPUTED ADDRESS IS = $2000, MEANING THAT
*  A PASS THROUGH THE MEMORY MAP HAS BEEN COMPLETED. OTHERWISE Z=0.
*
NXTADR   LDA    ADDR            GET MS.BYTE OF LAST ADDRESS USED
         BNE    NOT04F          BRANCH IF NOT IN PAGE ZERO EPROM AREA
         LDX    ADDR+1          GET LS.BYTE OF LAST ADDRESS USED
         CPX    #RAM-1          LAST BYTE OF PAGE ZERO EPROM?
         BNE    NOT04F          BRANCH IF NOT $004F
*
* MUST HAVE JUST ACCESSED LAST PAGE ZERO EPROM LOCATION IF HERE SO
* FORCE NEXT ADDRESS INCREMENT TO POINT TO MAIN EPROM AREA AT $0100.
*
         LDX    #$0100-1        POINT TO ADDRESS BELOW THE ONE WANTED
         STX    ADDR+1          PLACE IN LOCATION TO BE INCREMENTED
         STX    ADDRLO          PRESET PORTA PRIOR TO INCREMENT
*
* 16 BIT ADDRESS INCREMENT
*
NOT04F   INC    ADDR+1          INCREMENT LS. ADDRESS BYTE
         INC    ADDRLO          UPDATE LS.ADDRESS AT PORT A
         BNE    GOBACK          RETURN IF A PAGE BOUNDARY NOT REACHED
         INCA                   INCREMENT MS. ADDRESS
*
* LOOK OUT FOR HAVING GONE THROUGH THE ENTIRE MEMORY MAP.
*
         CMP    #$20            WAS THAT THE END OF MEMORY ( $1FFF )?
         BEQ    GOBACK          EXIT WITH Z=1 IF THE END WAS REACHED.
*
* LOOK OUT FOR HAVING ACCESSED THE LAST LOCATION IN THE MAIN BLOCK
*
         CMP    #$1F            WAS THAT THE END OF THE MAIN BLOCK
         BNE    INMAIN          BRANCH IF STILL WITHIN THE MAIN BLOCK
*
* SKIP OVER THE UNUSED AND BOOTSTRAP AREAS
*
         LDX    #$F0            FORCE LS. ADDRESS BYTE TO $F0
         STX    ADDR+1          .
         STX    ADDRLO          MAKE SURE PORT A GETS UPDATED AS WELL
```

Mask Set Errata 1
68HC705C8AMSE1
68HC705C8A 8-Bit Microcontroller Unit

INTRODUCTION

This errata provides information pertaining to security bit programming applicable to the following 68HC705C8A MCU mask set devices.

- 0E20T
- 1E20T
- 2E20T
- 2E79R
- 0E97N

MCU DEVICE MASK SET IDENTIFICATION

The mask set is identified by a 4-character code consisting of a letter, two numerical digits, and a letter (for example, E20T). Slight variations to the mask set identification code may result in an optional numerical digit preceding the standard 4-character code (for example, 2E20T).

MCU DEVICE DATE CODES

Device markings indicate the week of manufacture and the mask set used. The data is coded as four numerical digits where the first two digits indicate the year and the last two digits indicate the work week. The date code "9115" would indicate the 15th week of the year 1991.

MCU DEVICE PART NUMBER PREFIXES

Some MCU samples and devices are marked with an "SC" or "XC" prefix. An "SC" prefix denotes special/custom device. An "XC" prefix denotes device is tested but is not fully characterized or qualified over the full range of normal manufacturing process variations. After full characterization and qualification, devices will be marked with the "MC" prefix.

SECURITY BIT PROGRAMMING

When programming the security bit on the 68HC705C8A, it is required that either a higher programming voltage (V_{PP}) or a longer programming time be applied. Recommended programming time for masks covered in this document is 200 ms with a V_{PP} of 15 volts. Do not apply a V_{PP} of over 16.5 volts when programming. A series resistor is not needed for I_{PP} current limiting.

If using the evaluation module (EVM) board to program the security bit, V_{PP} should be set at 16.5 volts.

NOTE: Future revisions of the 68HC705C8A will require the normally specified programming time and voltage when programming the security bit.

Above recommendations apply only to the security bit programming and not to programming the EPROM array. Please use the normally specified programming voltage and time when programming the EPROM array.

Mask Set Errata 1

68HC705C8AMSE2 (R1)

68HC705C8A 8-Bit Microcontroller Unit

INTRODUCTION

This errata provides information pertaining to the extra I_{DD} current draw related to PORTD and is applicable to the following 68HC705C8A MCU mask set devices.

- 1E20T
- 2E20T
- 3E20T
- 2E79R
- 3E79R

MCU DEVICE MASK SET IDENTIFICATION

The mask set is identified by a 4-character code consisting of a letter, two numerical digits, and a letter (for example, E20T). Slight variations to the mask set identification code may result in an optional numerical digit preceding the standard 4-character code (for example, 2E20T).

MCU DEVICE DATE CODES

Device markings indicate the week of manufacture and the mask set. The data is coded as four numerical digits where the first two digits indicate the year and the last two digits indicate the work week. The date code "9115" would indicate the 15th week of the year 1991.

MCU DEVICE PART NUMBER PREFIXES

Some MCU samples and devices are marked with an "SC" or "XC" prefix. An "SC" prefix denotes special/custom device. An "XC" prefix denotes device is tested but is not fully characterized or qualified over the full range of normal manufacturing process variations. After full characterization and qualification, devices will be marked with the "MC" prefix.

ADDITIONAL I_{DD} CURRENT DRAW

When reading PORTD, a condition exists that causes additional I_{DD} current to be drawn . If any pin on PORTD transitions from a logic zero to a logic one, current through V_{DD} increases by approximately 300 µA. However, this does not affect pin leakage. Also, the actual read of PORTD is not affected. This condition cannot be cleared or reset, but powering the part down will return it to its initial condition.

The transition from logic zero to a logic one condition will cause a violation in the stop I_{DD} specification and may cause a violation in the wait I_{DD} specification depending on the frequency and voltage of operation. The operating I_{DD} will remain within specification.

For low-power or current-sensitive applications, it is recommended not to transition PORTD pins.

This condition will be fixed in future mask revisions of the 68HC705C8A. However, to make the MC68HC705C8A compatible with other C Family parts, the new mask revision will require that all input pins be tied to either a logic one or logic zero. The input pins should not be left floating.