MOTOROLA

TECHNICAL UPDATE

MC68HC05B4
MC68HC705B5
MC68HC05B6
MC68HC05B8
MC68HC05B16
MC68HC705B16
MC68HC05B32
MC68HC705B32

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

B Series General Information

COP (Computer Operating Properly) Watchdog Timer

COP1MISC

COP Watchdog Timer Revision History

Date	Revision	Description
5/20/94	1	Original release. Includes tracker HC05B4.001.
3/30/95	2	Reformatted

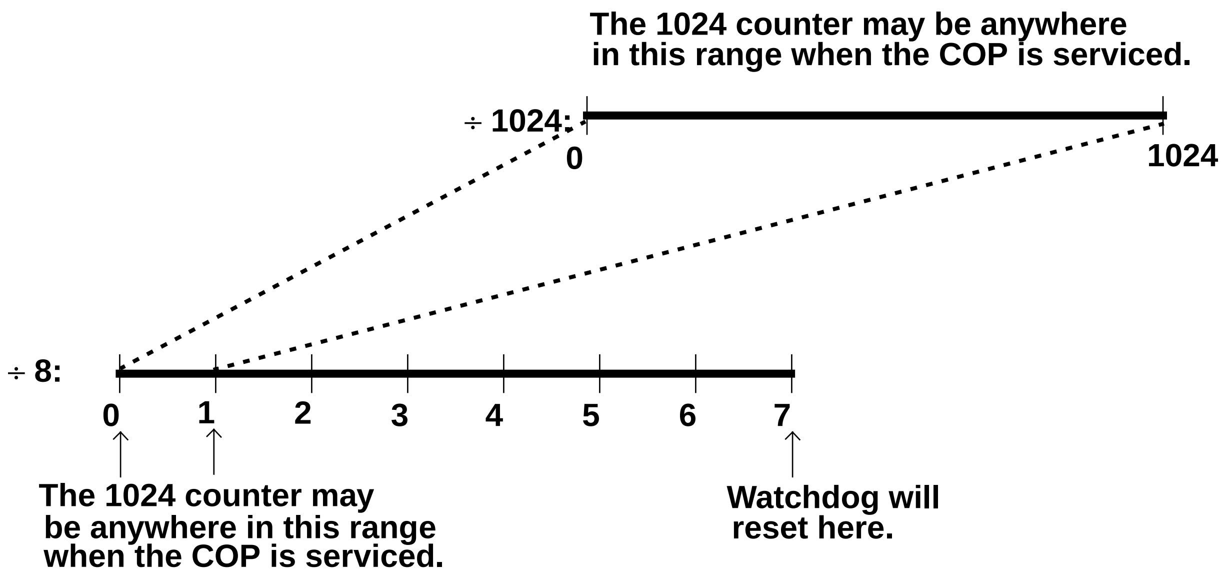
Watchdog Timeout Period

Tracker Number: HC05B4.001 Revision: 1.00
Reference Document: MC68HC05B6/D Rev. 3, page 9-3

Some customers have had problems relating the explanation of the COP timeout to the watchdog timeout spec.

Figure 9-2 shows that the watchdog timeout is generated by a divide-by-1024 counter and a divide-by-8 counter. From that figure, one might think that the minimum timeout is 7168 cycles (7 counts of 1024) and a maximum is 8192 cycles (8 counts of 1024.) The minimum and maximum counts result because the watchdog is reset somewhere between count 0 and count 1024 within the first count of the divide-by-8 counter. Refer to Figure 1 below. These results do not reflect the values given in the reference documents.

In the control timing table, the minimum watchdog timeout is stated as 6144 cycles and the maximum is 7168 cycles. Figure 9-2 is somewhat misleading. The divide-by-8 counter is a misnomer in that it has eight states from 0 to 7. When the divide-by-8 counter is reset, it is reset to 0. When the divide-by-8 counter reaches state 7, a timeout occurs. Therefore, the divide-by- 8 counter will count only six full cycles of 1024, giving a total count of 6144. These six cycles are 1 to 2, 2 to 3, 3 to 4, 4 to 5, 5 to 6, and 6 to7. After the 7th state is hit, the timeout occurs. In this manner, the timeout minimum is 6144 cycles and the maximum is 7168 cycles. Refer to Figure 1 below.


The 1024 counter may be anywhere in this range when the COP is serviced.
÷ 1024:
0
1024
÷ 8:
0
1
2
3
4
5
6
7
The 1024 counter may be anywhere in this range when the COP is serviced.
Watchdog will reset here.

timeout = (6 X 1024) + (0 to 1024) = 6144 to 7168 count

Figure 1. Divide-by-8 and Divide-by-1024 Counters

MC68HC05 CPU Core

HC05CPU_A

The HC05 CPU is the central processing unit that is common to all HC05 microcontrollers. This is the functional block that sets HC05 MCUs apart from others.

HC05CPU Core Revision History

Date	Revision	Description
6/15/94	1	Original Release. Includes trackers HC05CPU.001, HC705C8.017, HC705C8.018, HC705C8.019.
3/30/95	2	Reformatted

Correction to SUB in Applications Guide

Tracker Number: HC05CPU.001 Revision: 1.00
Reference Document: M68HC05 Applications Guide MC68HC05AG/AD Rev. 1, Page A-62

Replace the C bit description with this:

The C bit (carry flag) in the condition code register becomes set if the absolute value of the contents of memory is larger than the absolute value of the accumulator. Otherwise, if this is not the case, the C bit is cleared.

I Bit in CCR During Stop Mode

Tracker Number: HC705C8.017 Revision: 1.00
Reference Documents: M68HC05 Applications Guide MC68HC05AG/AD Rev. 1, Page 3-93

The flow chart for stop mode 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 can release the processor from stop mode.

I Bit in CCR During Wait Mode

Tracker Number: HC705C8.019 Revision: 1.00
Reference Document: M68HC05 Applications Guide MC68HC05AG/AD Rev. 1, Page 3-93

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

BSET and BCLR are RMW Instructions

Tracker Number: HC705C8.018 Revision: 2.00
Reference Document: HC05 Technical Data books

In many of the databooks, the table of read-modify-write (RMW) instructions in the instruction set and addressing mode section does not list the BSET and BCLR instructions. Rather, these databooks list BSET and BCLR only as bit manipulation instructions. While this is correct, it is not complete, however, since these operations use an RMW method to accomplish their task. The BSET and BCLR instructions, 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, which 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.

EEPROM1

EEPROMB256_A

EEPROM1 Revision History

Date	Rev	Description
5/20/94	A01	Original Release. Includes trackers HC705B16.002, HC705B16.008.
3/30/95	A02	Reformatted. Added EEPROM B256_A.

Programming the Options Register in the EEPROM1

Tracker Number: HC705B16.008   Revision: 1.00
Reference Document: MC68HC05B6/D Rev. 3, page 3-6

All HC05B Family devices (except the HC05B4 and HC705B5) have an EEPROM1 array. Address $100 is known as the option register. Bits 0 and 1 are used as the SECurity and EE1Protect bits, respectively. The remaining six bits are standard EEPROM cells and can be used to store data if needed. To reduce the possibility of error by securing the device unintentionally, it is recommended that these bits not be used.

Options Register (OPTR) Location Correction

Reference Document: MC68HC05B6/D Rev.3, page 3-6

Tracker Number: EEPROMB256_A.001   Revision: 1.00

The first sentence of this sectioncurrently reads:

"This register (OPTR), located at $0010, contains ....".

However, this is incorrect and should be replaced with:

"This register (OPTR), located at $0100, contains . . . "

V_{PP}1 Connections, Problems, and Gotchas

Tracker Number: HC705B16.002 Revision: 2.00

The following is further clarification on the V_{PP}1 pin:

NOTE: The EEPROM1 array and V_{PP}1 pin are common to all HC05B devices except for the HC05B4 and HC705B5. The HC(7)05X16 devices also have a V_{PP}1 pin and the EEPROM1 array.

The V_{PP}1 pin is the output of the charge pump for the V_{PP}1 array. The charge pump is a capacitor/diode ladder network, which gives a high voltage but with a low drive capability (for example, a very high impedance output of between 20 and 30 MΩ). The charge pump output is brought to the outside world (via the V_{PP}1 pin) for Motorola factory use only. The charge pump is capable of programming only one byte at a time.

Any leakage path to V_{SS} or V_{DD} on the V_{PP}1 pin can load the charge pump and not allow the pump to reach the needed programming voltage. This could result in the EEPROM byte not being erased or programmed properly. Typically, placing an oscilloscope probe of 10 MΩ impedance on the V_{SS}1 pin will be sufficent to cause this.

The bottom line is: If in-circuit programming of the EEPROM1 array is needed, the V_{PP}1 pin should be left unconnected regardless of the V_{DD}. This will ensure that no leakage path to the supplies exists. This pin should never be tied to V_{DD} or $V_{SS,}$ if the user wishes to program or erase the EEPROM1 in an application. If the user programs the EEPROM1 array before running a normal day-to-day applicatio, and if the user never writes or erases the EEPROM1 array, then the pin can be tied to V_{DD}.

Connecting V_{PP}1 to V_{DD} can be considered as a security feature because the EEPROM1 array is protected from being erased or programmed in runaway conditions. (The charge pump output has been clamped to $V_{DD).}$ Doing this, however, will increase the power consumption.

As stated previously, the V_{PP}1 pin is a very high impedance pin. This high impedance can cause problems with some applications. For example, very high humidity or very noisy environments can cause programming problems. The following is a quick look at what to do with V_{PP}1 in some of these situations.

For example, if the application does not write to or erase the EEPROM1 array during normal use, connect the V_{PP}1 pin to V_{DD}. This will ensure that no noise will influence the EEPROM circuit.

Also, if the application does write to or erase the EEPROM1 array during normal use, shield this pin as much as possible. Use plenty of GND track nearby. Some customers have placed a small capacitor (1 nF) to decouple the V_{PP}1 pin. This should not affect the programming in normal use (10 mS programming time) despite slightly lengthening the rise time of $V_{PP}1$.

Programmable Timer

TIM2IC20CF_A

Programmable Timer Revision History

Date	Revision	Description
5/20/94	1	Original Release. Includes tracker HC805B6.002.
3/30/95	2	Reformatted

Output Compares and Flags

Tracker Number: HC805B6.002 Revision: 1
Reference Documents: MC68HC05B6/D Rev. 3, page 5-9

This code snippet shows one method of setting up and distinguishing between output compares 1 and 2.

```
********************************************************************************
*
*
*
*
*
*
*
*       Program Name: OCB5.ASM
*
*       Revision:1.01
*
*       Date: August 11, 1992
*
*
*
*       Written By: Dan Bernard
*
*               Motorola CSIC Applications
*
*
*
*       Original Assembly: P&E Microcomputer Systems CASM05
*
*
*
*               ******************************
*
*               *       Revision History     *
*
*               ******************************
*
*       Revision:
*
*       1.00 8-11-92 Original Source Written with 705B5 memory map
*
*       1.01 4-26-93 Modified to run on HC805B6 EEPROM1
*
********************************************************************************
*
```

```
******************************************************************************
*
*       Program Description:
*
*
*
*       This program shows one method of handling the Interrupt enable bits
*
*       and flags for both OC1 and OC2  Timer Output Compare functions.
*
*       Originally written for use on the HC705B5
*
*       Also of interest are the 16-Bit addition routines
*
*
*
*       Start up conditions=
*
*
*
*       When the code first starts to run any pending timer interrupts are
*
*       cleared, the current value of the counter timer is retrieved. An offset
*
*       is added to it which represents the width of the pulse we want out.
*
*       This is done with a 16-Bit addition. These routines may prove useful
*
*       for other applications.  Do this for both output compare registers.
*
*       The timer interrupts are then enabled along with issuing a global
*
*       interrupt enable. An endless loop is then entered.  Other code can be
*
*       inserted at this point to do other things while waiting for an
*
*       interrupt.  See the comments in the individual routines for additional
*
*       information on the operation of this program.
*
******************************************************************************
*
```

```
**************Output Compare Register Equates****************

tcr     equ     $12             ;Timer Control Register
tsr     equ     $13             ;Timer Status Register
oc1h    equ     $16             ;Output Compare 1 High
oc1l    equ     $17             ;Output Compare 1 Low
tch     equ     $18             ;Timer Counter High
tcl     equ     $19             ;Timer Counter Low
atch    equ     $1a             ;Alternate Timer Counter High
atcl    equ     $1b             ;Alternate Timer Counter Low
oc2h    equ     $1e             ;Output Compare 2 High
oc2l    equ     $1f             ;Output Compare 2 Low

*********Bit Equates***********************************

*********TCR***************

folv2   equ     $10             ;Force output compare 2
folv1   equ     $08             ;Force output compare 1
olv2    equ     $04             ;OC2 output level
olv1    equ     $01             ;OC1 output level
ocie    equ     $40             ;Output compare interrupt enable
folv2b  equ     $04             ;bit 4 of tcr another name for folv2
folv1b  equ     $03             ;bit 3 of tcr another name for folv1
olv2b   equ     $02             ;bit 2 of tcr another name for olv2
olv1b   equ     $00             ;bit 0 of tcr another name for olv1
ocieb   equ     $06             ;bit 6 of tcr another name for ocie

*********TSR***************

ocf1    equ     $40             ;OC1 output compare flag
ocf2    equ     $08             ;OC2 output compare flag
ocf1b   equ     $06             ;bit 6 of tsr another name for ocf1
ocf2b   equ     $03             ;bit 3 of tsr another name for ocf2

*********Vector Equates********

oci     equ     $1ff6           ;OC interrupt vector location
reset   equ     $1ffe           ;Reset vector location

*********Memory Equates*******

ram     equ     $050            ;Beginning of ram
eeprom1 equ     $101            ;805b6 eeprom1
eeprom6 equ     $0800           ;805b6 eeprom6
rom     equ     $800            ;Beginning of main rom area

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

```
*********Variables*****************************************

       org     ram            ;start of ram variable area
acch   ds      1              ;high byte of 16 bit accumulator
accl   ds      1              ;low byte of 16 bit accumulator
temph  ds      1              ;high and low byte temp locations
templ  ds      1              ;for value to be added to acch
                              ;and accl

*********Program Start*************************************

       org     eeprom1        ;program starting address

*********Initilize Stuff********

init:
       lda     tsr            ;do first part of clearing flags
       ldx     #tch           ;get address of timer counter
       jsr     load_acc       ;and load 16 bit accumulator
       ldx     #temph         ;set up to point to second operant
       lda     #$00           ;this is the high byte pick what
       sta     0,x            ;value you want.
       lda     #$ff           ;this is the low byte of sec operant
       sta     1,x            ;store it
       jsr     add_16         ;adds the two numbers in the 16 bit acc
                             ;and the temp variable location
       ldx     oc1h           ;point to OC1 register
       jsr     store_res      ;store addition to the OC1 register
       ldx     oc2h           ;point to OC2 register
       jsr     store_res      ;store addition to OC2 register
                              ;This gives us a common starting point
       bset    ocieb,tcr      ;enable timer interrupts
       cli                    ;globally enable interrupts

********Main Loop***********

main_loop:
       bra     *              ;do other stuff here or spin wheels
       nop                    ;waiting for an interrupt
       nop                    ;these nop's do nothing they are here
                              ;to help the debugger from getting confused

*********Interrupt Handler*********************************************
*        This routine is entered when one of the Output*
*        Compares has a sucessful compare. We first check to*
*        see if OC1 generated the intertie if not we test if*
*        OC2 did it. From there we change the Output Compare*
*        Register so that we can generate a square wave on*
*        the output pins*
*******************************************************************
```

```
outcmp:                     ;main entry point
       lda    tsr           ;get status ck if OC1 caused interrupt
       bit    #ocf1
       bne    oc1_1         ;go to OC1 routine if so other wise
       and    #ocf2         ;ck to see if OC2 caused the interrupt
       bne    oc2_2
       bra    exit1         ;it must have been an interrupt from
                            ;the TOF or Input Capture
oc1_1:
       ldx    oc1h          ;point to OC1 register
       jsr    load_acc      ;put data in 16 bit acc
       ldx    #temph        ;point to temp variable
       lda    #$d0          ;this is the high byte of offset
       sta    0,x           ;store it
       lda    #$00          ;this is the low byte of offset
       sta    1,x           ;store it
       jsr    add_16        ;do the addition
       ldx    #oc1h         ;point to OC1 register
       jsr    store_res     ;store to OC1 register
       brset  olv1b,tcr,invert      ;check current state of output level
       bset   olv1b,tcr     ;it must have been a zero
       lda    tsr
       and    #ocf2         ;before we go let's test OC2 just in
       bne    oc2_2         ;case
       bra    exit1         ;time to go back
invert:
       bclr   olv1b,tcr     ;it must have been a one
       lda    tsr
       and    #ocf2         ;Ck OC2 before leaving
       bne    oc2_2
       bra    exit1

oc2_2:                      ;if we are here OC2 must have caused
       ldx    oc2h          ;point to OC2 register
       jsr    load_acc      ;put data in 16 bit acc
       ldx    #temph        ;point to temp variable
       lda    #$ff          ;this is the high byte of offset
       sta    0,x           ;store it
       lda    #$10          ;this is the low byte of offset
       sta    1,x           ;store it
       jsr    add_16        ;do the addition
       ldx    #oc2h         ;point to OC2 register
       jsr    store_res     ;store to OC2 register
       brset  olv2b,tcr,invert2     ;check current state of output level
       bset   olv2b,tcr     ;must have been a zero
       bra    exit1         ;we're done
invert2:
       bclr   olv2b,tcr     ;it must have been a one
       bra    exit1         ;we're done
exit1:
       rti                  ;the end of interrupt routine
```

```
*********Sixteen Bit Addition routines***************************************

******************************************************************
*       load_acc                                                 *
*       routine to load 16 bit psuedo acc                        *
*       x index register contains source address                 *
*       of data msb at low address                               *
*       16 bit accumulator consists of two ram                   *
*       locations "acch" and "accl"                              *
******************************************************************

load_acc:
        lda     0,x             ;load msb of data
        sta     acch            ;store in msb of accumulator
        lda     1,x             ;get lsb of data
        sta     accl            ;store in lsb of accumulator
        rts
******************************************************************
*       store_res                                                *
*       after 16 bit math this routine stores resultant data in  *
*       accumulator to the memory location pointed to by the     *
*       x index register. The x register points to the msb of    *
*       data                                                     *
******************************************************************

store_res:
        lda     acch            ;get high byte of data
        sta     0,x             ;store it
        lda     accl            ;get low byte of data
        sta     1,x             ;store it
        rts
******************************************************************
*       add_16                                                   *
*       acch and accl contain first operant on entry and         *
*       contains the 16 bit result on exit. The x index          *
*       register points to the second operant of the 16 bit      *
*       addition. This would normally be the temp variable       *
******************************************************************

add_16:
        lda     accl            ;get low byte of first operant
        add     1,x             ;add to second operand
        sta     accl            ;and store in back in accl
        lda     acch            ;get high byte of first operant
        adc     0,x             ;add high bytes together + carry
        sta     acch            ;from low byte add
        rts
```

```
*********Vector Assignments************

       org    oci           ;Output Compare Vector location
       dw     outcmp        ;routine address
       org    reset         ;Reset Vector location
       dw     init          ;program start address
end:
```

Packaging Types

52-Pin PLCC Pinout

Date	Revision	Description
4/20/95	1	Original Release. Includes tracker HC05B6.002.

Tracker Number: HC05B6.002 **Revision: 1.00**

Reference Document: MC68HC05B6/D Rev. 3, page 12-1

Include the following table:

Device	Pin 15	Pin 40
MC68HC05B4	NC	NC
MC68HC05B6 MC68HC05B8 MC68HC05B16 MC68HC05B32	NC	$V_{PP}1$
MC68HC705B5	$V_{PP}6$	NC
MC68HC705B16 MC68HC705B32	$V_{PP}6$	$V_{PP}1$

The NC pin 6 is always NC.

HC05 B Series Part Specific

MC68HC705B5

Date	Rev	Description
4/7/95	1	Original Release. Includes trackers HC705B5.005 and HC705B5.001
4/17/95	2	

Security

Tracker Number: HC705B5.001 Revision: 1.00

Reference Document: Not applicable

The security bit currently is not available on the HC705B5. The security fuses are experimental. They will be available to the user only after being characterized by product engineering.

When programming the option register on the 705B5, make sure the SECE bit is cleared. In other words, bit 7 of location $1EFE should be a zero.

Do not program the SECE bit for any reason.

Bootloader for Mask Sets 0D10J, 2B40T, 4B40T

Reference Document: Not applicable

Tracker Number: HC705B5.008 Revision: 1.00

Mask Set: 0D10J, 2B40T, 4B40T

```
*********************************************************************
*
*
*       BOOTSTRAP LOADER for the MC68HC705B5
*       ====================================
*
*       Rev   C.01           1989/Nov/15  SPP
*
```

```
*       Programmer :   O. Pilloud
*       Updates    :   M. Bron  MOTOROLA GVA
*                      C. Humbert
*
*
*                       -----------
*                      ( BOOTSTRAP )
*                       -----------
*                           / \
*                          /   \ active
*                         < SEC ? >---------> ( flash red led )
*                          \   /
*                           \ /
*                            : no
*                           / \
*                       1 /   \
*          +-----------------< PD4 ? >
*          :               \   /
*          :                \ /
*         / \                :0
*        /   \ 1             :
*       < PD3 ? >----+        :
*        \   /   :           :
*         \ /    :           :
*          : 0   :           :
*          :     :           :
*      -----------   :           :
*      :  RAM    :   :        / \        / \
*      :  BOOT   :   :       /   \ 1    /   \ 1
*      :  LOADER :   :      < PD3 ? >----< PD2 ? >-------+
*      -----------   :       \   /      \   /        :
*                :        \ /        \ /        V
*          +--------+        0 :        0 :   -------------------
*          V                 V          :  ( EPROM Gate stress )
*      -----------        ------------       :   -------------------
*      :  SERIAL :        : PARALLEL :      V
*      : ROUTINES:        :  PROG &  :     -------------------
*      :        :        : VERIFY   :   ( EPROM erase check )
*      -----------        ------------     -------------------
*
* NOTE : This flowchart is intended only as a mean to outline the
*        possible program paths, it is not an exact reflection of
*        all the  details of program dispatching and routing.
*
*******************************************************************
*
*
* Bootstrap loading can be done either through a parallel port
* using an external EPROM or from any other parallel source,
* using the handshake facility provided.
*
* Bootstrap loading can also be accomplished through the SCI
* in serial form, at 9600 baud for a nominal crystal frequency
* of 4 MHz. The mode selection is as follows :
*
*
*
*               PORT D4 : PORT D3
*               --------+--------
*                  0    :    0      EPROM Bootstrap parallel
*                  0    :    1      EPROM erase check
*                  1    :    0      RAM Bootloader
*                  1    :    1      Serial Bootstrap
*
*
* Dump or serial load is accomplished through the same routine,
* and both functions share a common protocol, as follows :
*
*
*
* - 1) B5 sends back the 4 bytes just programmed as a prompt and also
*      for checking by the host (the first and second time this data is
*      irrelevant).
*
```

```
* - 2) B5 expects 6 bytes from HOST :
*          : ADDRESS HI:LO : DATA(a):DATA(a+1):DATA(a+2):DATA(a+3) :
*
* - 3) B5 takes a nominal 5 ms to programm the data, while the next 6
*      bytes are being transmitted as outlined above.
*
* - 4) Loop to 1)
*
*
* Note that the program implements a pipeline structure in that 3
* operations take place simultaneously :
*
* - echo bytes are sent back to the host
* - data is being programmed
* - next data is being received
*
* It is possible to load the RAM (above address XADR)
* with a test program and execute it. Execution  is triggered
* by sending a negative (bit 7 set) high address byte. Execution
* starts at XADR.
*
* A mechanism is provided in both the serial and parallel case to blow
* the security fuse if required.
*
*
*        I/O and INTERNAL registers definitions

*
            ORG 0
*
PORTA    RMB   1          port A
PORTB    RMB   1          port B
PORTC    RMB   1          port C
PORTD    RMB   1          port D
DDRA     RMB   1          port A DDR
DDRB     RMB   1          port B DDR
DDRC     RMB   1          port C DDR
```

```
*
ECONT    RMB   1              EPROM control register
ADR      RMB   1              A/D data register
ADC      RMB   1              A/D status and control register
PLMA     RMB   1              pulse length mod reg A
PLMB     RMB   1              pulse length mod reg B
MISC     RMB   1              miscellaneous register
BAUD     RMB   1              SCI baud
SCCR1    RMB   1              SCI control register 1
SCCR2    RMB   1              SCI control register 2
SCSR     RMB   1              SCI status register
SCDAT    RMB   1              SCI data register
TIMC     RMB   1              TIMER control register
TIMST    RMB   1              TIMER status register
ICR1     RMB   2              capture register 1 (16 BIT)
OCR1     RMB   2              output compare register 1 (16 BIT)
TIMER    RMB   2              TIMER free running counter (16 BIT)
DUALTM   RMB   2              alternate counter register (16 BIT)
OCR2     RMB   2              output capture register 2 (16 BIT)
COMP2    RMB   2              compare register 2 (16 BIT)
*
*
*       MEMORY MAP DEFINITIONS
*
*  Note : TEST is a write only register and its access is
*       authorized only when ELAT is cleared. When ELAT
*       is set, address $20 (P0EP6) is accessed for writing.
*
*
TEST     EQU   $20            TEST register
P0EP6    EQU   $20            page 0 EPROM 6 start address
RAM      EQU   $50            RAM start address
STACK    EQU   $FF            top of STACK
EPROM1   EQU   $100           EPROM 1 start address
XROM     EQU   $200           Extra ROM
EPROM6   EQU   $800           EPROM 6 start address
OPTR     EQU   $1EFE          OPTION reg
```

```
BOOTR     EQU    $1F00             BOOTSTRAP ROM start address
BUTVCT    EQU    $1FE0             BOOTSTRAP ROM vectors
VECT      EQU    $1FF0             EPROM6 vectors
*
*
*         Miscellaneous definitions and equates
*
*
HI        EQU    0                 hi byte offset
LO        EQU    1                 lo byte offset
MS20      EQU    38                20 ms timing factor
GSTR      EQU    %01100000         XCOL and XROW at 1.
LONG      EQU    $09               long timing factor (5 ms nominal)
SHORT     EQU    $01               short timing factor (.85 ms nominal)
RED       EQU    PLMA              red LED on PLMA
GREEN     EQU    PLMB              green LED on PLMB
OCF1      EQU    6                 output compare 1 flag
EPGM      EQU    4                 ECONT  bit definition
ELAT      EQU    5                 ECONT
FUSE      EQU    7                 ECONT    1 = secure
IN        EQU    6                 Handshake IN line on Port C
OUT       EQU    5                 Handshake OUT line on Port C
VPP       EQU    7                 VPP bit in portC
RDRF      EQU    5                 Receive data ready flag in SCSR
TDRE      EQU    7                 Transmit DATA Reg Empty
MBIT      EQU    4                 8 data bits flag in SCCR1
ADON      EQU    5                 A/D converter control bit
TOF       EQU    5                 Timer overflow flag
OLVL1     EQU    0                 Output level 1
FOLV1     EQU    3                 Force output compare 1
*
*
*
*
*         Synthesized instructions
*
*  Note : In order to circumvent a shortcoming inherent to any
*       2 pass assembler as the one used here, some DIRECT
*       (JSR/LDA/STA/DEC/INC) instructions have been synthesized
*       using 2 FCB instructions. This is flagged by a @@@ in
*       the comment line.
*
*
LOADA    EQU   $B6           Load A direct
STORA    EQU   $B7           Store A direct
GOTO     EQU   $BC           Jump direct
CALL     EQU   $BD           Jump to subroutine direct
DECD     EQU   $3A           Decrement direct
INCD     EQU   $3C           Increment direct
LDAX1    EQU   $E6           Load A indexed, 1 byte
STAX1    EQU   $E7           Store A indexed, 1 byte
CMPX1    EQU   $E1           Compare indexed, 1 byte
*
*
*
*
*       PROGRAM Section
*       ===============
*
         ORG BOOTR
*
*
* RFLASH: Flash RED LED at about 4 Hz if security fuse blown.
*
RFLASH   COM    RED              toggle LED
         BRCLR  TOF,TIMST,*      Wait for timer overflow
         LDA    TIMER+LO         Clear TOF
         BRA    RFLASH           loop
*
*

*
*
* RAM BOOTSTRAP
* =============
*
*  Note : Transfer a program from an external EPROM into RAM.
*         Execution starts as soon as the RAM is full
*         (176 bytes transfered), or when PORTD4 goes low.
*
*
BUTRAM    BRCLR  4,PORTD,EXERAM     short load
          STX    PORTB              address EPROM
          BCLR   OUT,PORTC          handshake, ready
          JSR    GETPAR             get data, with
*                                   handshake
          STA    ,X                 put to RAM
          INCX
          BNE    BUTRAM             (FF => 00) = end
EXERAM    JMP    RAM                execute in RAM.
*
*
*
*
* ENTRY POINT TO BOOTSTRAP LOADER
* ==============================
*
*
*  Note : Port A is the DATA input for MCU programming.
*         Port B and the 5 lower bits of port C reflect the
*         address being programmed. Port C, bits 5 & 6 provide
*         handshake capability while programming the MCU.
*         If the handshake is not used, ie: in case DATA
*         comes from an external EPROM, these 2 pins
*         should be tied together. Pin PORTC7 is used to switch
*         Vpp on during programming.
*         Program routing (dispatch) is accomplished by the state
*         of pins PORTD3 & PORTD4 (Check flowchart for details).
*         Led's are driven from the PLM's and will be ON when
*         the PLM's are at $00 and (mostly) OFF when they hold an
*         $FF value.
```

```
*
*
*
ENTRY     BSET   OLVL1,TIMC        Tell the world, this is a B5
          BSET   FOLV1,TIMC        by setting TCMP1
*
SETIO     LDA    #%00100000        init PORTC, hi byte of address
          STA    PORTC             handshake, and Vpp off.
          COM    RED               red led off
          COM    GREEN             green led off
*
          BRSET  FUSE,ECONT,RFLASH  check SEC bit
          LDX    #RAM              for both SCINIT & BUTRAM
*
* now let's go as fast as possible to the RAM loader
* if requested to do so.
*
          BRSET  3,PORTD,SCINIT    branch if not parallel load
*
          COM    DDRB              Port B outputs
          LDA    #%10111111        IN handshake on port C
          STA    DDRC
*
          BRSET  4,PORTD,BUTRAM    ready for RAM load and exec
*
*
*  Initialise the SCI
*
SCINIT    BCLR   MBIT,SCCR1        8 data bits
          LDA    #%11000000        baud rate 9600
          STA    BAUD
          LDA    #%00001100        TE / RE
          STA    SCCR2             end of init
          STA    SCSR              clear TDRE & TC bits
*
*  Note : Some routines, while programming, must be executing
*        from RAM, so let's start by copying them form XROM
*       to RAM. The extended addressing routine BPS must also
*       reside in RAM.
*
*
COPYL   LDA   RAMR-RAM,X      transfer part of ROM to RAM and
        STA   ,X              thus erase it. X has previously been
*                             set to point to the bottom of RAM.
        INCX
        BNE   COPYL           until 176 bytes done ($FF - $50)
*
*
        BRSET  4,PORTD,SERIAL     serial load
        BRCLR  3,PORTD,PARINIT    parallel EPROM prog
        BRSET  2,PORTD,GSTRESS    EPROM gate stress setup
*
* Fall into ECHECK
*
* ECHECK: EPROM erase check. Read the whole array, using TABLE, and
*       check for zero.
*
ECHECK  CLRX
        FCB   CALL            @@@
        FCB   SSADD-OFST      Init base addr,
*                             in RAM to save bytes
MOREC   FCB   CALL            @@@
        FCB   BPS-OFST        get byte
        BNE   REDR            if not equal zero
        FCB   CALL            @@@
        FCB   BUMP-OFST       next address
        BNE   MOREC
        BRA   GREENR          all A OK
*
*
* GSTRESS:  Will put the EPROM in Gate stress mode for
*         reliability tests.
*
GSTRESS  FCB   GOTO           @@@
```

```
        FCB   GSTRAM-OFST      Continue in RAM
*
*
* PARINIT:  Parallel EPROM load. Init is in ROM, then jump in
*         RAM copy.
*
PARINIT   BRSET  2,PORTD,PARSH     Program time select
        LDA   #LONG           Default
        FCB   STORA           @@@
        FCB   DELAYT-OFST      In RAM
PARSH    LDA   #$C7            (STA| extended
        FCB   STORA           @@@
        FCB   BPS-OFST         set up BPS
*
        CLRX                  init address table pointer
        FCB   CALL            @@@
        FCB   SSADD-OFST       Init base addr,
*                             in RAM to save bytes
        FCB   GOTO            @@@
        FCB   PARPROG-OFST     program, from RAM.
*
*
*
* SCRD :  Routine SCRD services the SCI, it does that by polling
*       the RDRF (received data ready flag). It returns with
*       the byte of data in ACCA.
*
*
SCRD     BRCLR  RDRF,SCSR,*      Possibly wait for char
GDATA    LDA   SCDAT           get data & clear RDRF
        RTS
*
*
* SCWR :  Routine SCWR services the SCI, it does that by polling
*       the TDRE (transmit data register empty). It sends the
*       byte of data in ACCA over the SCI link.
*
```

```
SCWR      BRCLR  TDRE,SCSR,*        Wait for previous transmission
          STA    SCDAT
          RTS
*
*
*
*
* SERIAL: this is the MAIN serial (SCI) routine. It implements
*         the basic serial protocol. It is divided over both
*         BOOTROM and XROM, for ROM space usage purpose only.
*
*
SERIAL    BSET   7,DDRC             enable Vpp6 switch control
*
* ECHO & PROG bytes are initialized by COPYL
* (this is the first part of the serial loader loop)
*
SCILP     FCB    LOADA              @@@
          FCB    ECHO-OFST          First echo byte
          BSR    SCWR               send it
          FCB    CALL               @@@
          FCB    PROG08-OFST        first programming lap
          FCB    LOADA              @@@
          FCB    ECHO+1-OFST
          BSR    SCWR               send second echo byte
          FCB    CALL               @@@
          FCB    PROG08-OFST        second programming lap
          BSR    SCRD               get new address hi byte
          FCB    STORA              @@@
          FCB    NEWAD+HI-OFST      save it
          FCB    LOADA              @@@
          FCB    ECHO+2-OFST        third echo byte
          BSR    SCWR
          FCB    CALL               @@@
          FCB    PROG08-OFST        third programming lap
          BSR    SCRD               get new address lo byte
          FCB    STORA              @@@
```

```
        FCB   NEWAD+LO-OFST     save it too
        FCB   LOADA           @@@
        FCB   ECHO+3-OFST
        BSR   SCWR            last echo byte
        FCB   CALL            @@@
        FCB   PROG08-OFST       fourth programming lap
        BSR   SCRD            get first (lower addressed) data byte
        FCB   STORA           @@@
        FCB   RX+0-OFST         save it
* Free CPU time in RAM
        FCB   CALL            @@@
        FCB   PROG08-OFST       fifth programming lap
        BSR   SCRD            get second data byte
        FCB   STORA           @@@
        FCB   RX+1-OFST         save it
* Second free CPU time in RAM
        FCB   CALL            @@@
        FCB   PROG08-OFST       sixth & last programming lap
        BSR   SCRD
        FCB   STORA           @@@
        FCB   RX+2-OFST         get & save third data byte
*
REREAD  LDA   #$D6            LDA IX2
        FCB   STORA           @@@
        FCB   BPS-OFST          set up BPS as LDA indexed 16 bits
        LDX   #$3             reread what has just been prog'd
RLOOP   FCB   CALL            @@@
        FCB   BPS-OFST
        FCB   STAX1           @@@
        FCB   ECHO-OFST         and save as next echo
        DECX
        BPL   RLOOP
        JMP   CHKPRG
*
*
REDR    CLRA
        STA   RED
```

```
         DECA
         STA    GREEN
         BRA    *
*
GREENR    CLRA
         STA    GREEN
         DECA
         STA    RED
         BRA    *
*
*
*  BLOW:  If bit 7 at the EPROM option address $1EFE is
*         programmed, the fuse blowing starts with pulses
*         of 20 ms until the fuse is blown.
*
*
BLOW      BSET   7,BPA+LO-OFST      do not affect green led (DELAYV)
         LDA    OPTR
         BPL    NOBLOW             no need to blow
MBLO      BSET   FUSE,ECONT
         LDA    #MS20              20 ms
         FCB    CALL               @@@
         FCB    DELAYV-OFST
         BRCLR  FUSE,ECONT,MBLO    not blown yet
NOBLOW    RTS
*
*
* =========================================================
*                ORG XROM
*
*
*  Note : Routines after the label RAMR, will be copied to
*         RAM upon initialisation. They must execute in RAM,
*         as the ROM is not available when the EPROM latch
*         bit (ELAT) is set.
*
*
```

```
*
*
* EXEC :  Gives a possibility to load and execute a program
*         in high RAM from the SCI. The main SCI loader will
*         jump here if the high address byte is ever negative.
*
*
EXEC     JSR    BLOW             if required
         CLR    GREEN
         FCB    $BC              JMP instruction pointing to
         FCB    XADR-OFST        RAM, for SCI loader execution
*
*
* (this is the second part of the serial loader loop)
*
CHKPRG   LDX    #$3              compare with what has been prog'd
CLOOP    FCB    CALL             @@@
         FCB    BPS-OFST
         FCB    CMPX1            @@@
         FCB    PROG-OFST
         BEQ    NOERR
         CLR    RED              if error detected
NOERR    DECX
         BPL    CLOOP
*
MOVE     FCB    LOADA            @@@
         FCB    RX+2-OFST        transfer RX bytes as next PROG bytes
         FCB    STORA            @@@
         FCB    PROG+2-OFST      except fourth one not received yet
         FCB    LOADA            @@@
         FCB    RX+1-OFST
         FCB    STORA            @@@
         FCB    PROG+1-OFST
         FCB    LOADA            @@@
         FCB    RX-OFST
         FCB    STORA            @@@
         FCB    PROG-OFST
```

```
        FCB   LOADA          @@@
        FCB   NEWAD+LO-OFST    transfer address
        FCB   STORA          @@@
        FCB   BPA+LO-OFST
        FCB   LOADA          @@@
        FCB   NEWAD+HI-OFST
        FCB   STORA          @@@
        FCB   BPA+HI-OFST
*
        JSR   SCRD           get last data byte and put it
        FCB   STORA           @@@
        FCB   PROG+3-OFST      directly in place
*
        TST   BPA+HI-OFST      is hi address negative ?
        BMI   EXEC           if so exec
        JMP   SCILP          else go on loading
*
*
* VERFP : Verify programming, after PROG6.
*       The content of both EPROM's will be verified.
*       This routine can execute in ROM.
*
*       After PROG6, BPS contains $C7 (STA Extended)
*
VERFP   LDA   #$FF
        STA   GREEN           green LED off
        FCB   INCD           @@@
        FCB   BPS-OFST         init BPS $C7 => $C8 (EOR)
*
        CLRX                 init address table pointer
        BSR   SSADD           set start address
*
MOREV   BSR   XSFER           put address to ports and get data
*
        FCB   CALL           @@@
        FCB   BPS-OFST         compare (A| with EPROM6
        BNE   NOVERF          humm.. no good
        BSR    BUMP             next address
        BNE    MOREV              more to go ...
*
        JSR    BLOW             blow fuse if required
        JMP    GREENR
*
*
NOVERF    JMP    REDR              Red LED on, green LED already off
*
*
*
*  Note : The following routines will be executing in RAM,
*         being transfered by the BOOTSTRAP LOADER. Some data
*         is also initialized at the same time.
*
*
RAMR      EQU     *                start of routines to be copied
*                               in RAM upon initialisation.
*
* Offset due to the moving into RAM of some XROM routines :
*
OFST      EQU    RAMR-RAM
*
* NOTE :  The bytes affected (as far as assembly goes) by the
*         move from XROM to RAM are "corrected" by subtracting
*         OFST. This way, the generated addresses will be what
*         is required for execution in RAM. As far as reading
*         the listing is concerned, this offset can be ignored.
*
* BPS/BPA routine; this routine, residing in RAM, is used to
*         access the entire memory space. It is modified by
*         software (sorry about that folks), and has the form :
*
*
*         : EXTENDED INSTR : 16 BIT OPERAND : RTS :
*
*
```

```
*         BPA is initialized to PROG so that on the first loop
*         of the serial loader, (dummy) programmed data does not
*         provoque an error check. It follows that the last two
*         records transmitted dummies, made up of 00 00 00 00's.
*
BPS       FCB    $C6              (LDA| 16 bit-extended
BPA       FDB    PROG              address counter/argument
          RTS                 init to non EPROM addr for SCILP
*
*
*
*
*         Delay Routine
*
*         If entered at DELAYL, the timing will be 5 ms
*         If entered at DELAY, ACCA must hold a value of $01 (SHORT)
*         for a delay of 850 us.
*         The delay routine switches EPGM on upon entry
*         except if entered at DELAYV.
*
DELAY     FCB    LOADA            @@@
          FCB    DELAYT-OFST      From RAM
          BSET   EPGM,ECONT       start programming
*
DELAYV    BSET   VPP,PORTC        Turn Vpp on
*
          STA    TIMER+LO         reset timer
          STA    OCR1+HI          + put OCR1H
          STA    TIMST          :   first part of clear timer status
          LDA    #$C0           timer lo byte
          STA    OCR1+LO          + put OCR1L and 2nd part of clr tim stat
*
          BRCLR  OCF1,TIMST,*     wait for preset delay
*
          BCLR   VPP,PORTC        Turn Vpp off
          CLR    ECONT            end programming
          BRCLR  7,BPA+LO-OFST,*+3  Carry reflects bit tested
```

```
        ROR    GREEN            Flash green LED at abt 3 Hz
*
        RTS
*
*
* PROG08: Program serially in little bursts of 850 us interleaved
*       between SCI services, so as to save time.
*
PROG08    LDA     #$D7              STA IX2
          FCB     STORA             @@@
          FCB     BPS-OFST          set up BPS as STA indexed 16 bits
          BSET    ELAT,ECONT
          LDX     #$3               write 4 bytes every time
PLOOP     FCB     LDAX1             @@@
          FCB     PROG-OFST
          BSR     BPS
          DECX
          BPL     PLOOP
          BSR     DELAY
          RTS
*
*
*  Note : None of the routines beyond this point will be
*       needed by the SCI loader. So if a program is
*       loaded in RAM, by the SCI loader, it can start
*       at this address.
*
XADR      EQU     *
*
*
*       XSFER Routine
*
*  Note : XSFER (transfer) gets the current address from BPA,
*       puts it out to the ports, takes care of the handshake
*       protocol, and finally gets the data. If the handshake
*       is not used, connecting together IN with OUT (resp.
*       PORTA 6 & 5) will in effect disable it.
*       Note that if it is required, the OUT line (active
*       low) can be used as an Output Enable line to an
*       external EPROM.
*
*  Note : In the emulator, the end of the erase operation is
*       detected by monitoring the first falling edge of
*       the handshake OUT (PORTC5) line.
*
XSFER    FCB   LOADA          @@@
         FCB   BPA+LO-OFST     get low byte of address
         STA   PORTB          put it out
         FCB   LOADA          @@@
         FCB   BPA+HI-OFST     get high byte of address
         STA   PORTC          put it out & set data request
*
GETPAR   BRSET  IN,PORTC,*     wait for data valid
*
         LDA   PORTA          get data
         BSET  OUT,PORTC       acknowledge data
         RTS                  (A| holds data
*
*
*       BUMP/SSADD Routine
*
*  Note : The routine must first be entered at SSADD (Set Start
*       ADDress), with X cleared. It will initialize from TABLE
*       the address for routine BPS (which accesses memory), and
*       leave the index pointing to the next table entry, that
*       is the end of the current memory segment.
*
*       When subsequently entered through BUMP, BPS address
*       (at BPA) will be incremented by one, until it is equal
*       to the current "end of memory segment" pointed to
*       by X. Then the index is incremented, to point to the
*       start address of the next segment, and control goes
*       to SSADD to initialize the address of the next segment.
*
```

```
*       Upon exit from SSADD/BUMP, the Z flag is always cleared,
*       except iff the end of table has been reached : in that
*       case only, Z is set, signaling that the whole memory
*       array has been scanned.
*
BUMP    FCB   INCD            @@@
        FCB   BPA+LO-OFST       bump lo byte
        BNE   NOC             no carry
        FCB   INCD
        FCB   BPA+HI-OFST       carry over to high byte
*
NOC     FCB   LOADA           @@@
        FCB   BPA+HI-OFST       check high byte first
        FCB   CMPX1           @@@
        FCB   TABLE+HI-OFST
        BNE   GOON            no match yet
        FCB   LOADA           @@@
        FCB   BPA+LO-OFST       check low byte then
        FCB   CMPX1           @@@
        FCB   TABLE+LO-OFST
        BNE   GOON            no match
*
        CPX   #LAST-TABLE       end of table ?
        BEQ   GOON            Z set upon exit
RETRY   INCX
        INCX                  next table entry
*
SSADD   FCB   LDAX1           @@@
        FCB   TABLE-OFST        set high byte of address
        FCB   STORA           @@@
        FCB   BPA+HI-OFST
        INCX
        FCB   LDAX1           @@@
        FCB   TABLE-OFST        low byte of address
        FCB   STORA           @@@
        FCB   BPA+LO-OFST
        INCX
```

```
*
GOON      RTS                    Z set only if end of memory
*
*
*
* PARPROG: Will program all EPROM, by 4 bytes, from an
*         external program source. DATA is fetched from
*         PORT A, with or without handshake (see XSFER).
*
*
PARPROG   EQU    *
*
MOREP     LDA    #$4
          FCB    STORA             @@@
          FCB    COUNTER-OFST
*
          BSET   ELAT,ECONT
*
MOREL     BSR    XSFER             put address to port and get data
          BSR    BPS               store in EPROM latches
          BSR    BUMP              next address
          BEQ    LASTP
          FCB    DECD              @@@
          FCB    COUNTER-OFST
          BNE    MOREL
*
          BSR    DELAY             go program
          BRA    MOREP             more to go ...
*
LASTP     BSR    DELAY
          JMP    VERFP             verify, in ROM.
*
*
* GSTRAM: Sets the EPROM in Gate stress mode with data bytes
*         at $00,$00,$00,$00, and holds on with both LED's off.
*
GSTRAM    LDA    #GSTR             R/M/W not possible on $0020
```

```
         STA    TEST              OK in BOOT and NUM modes
         BSET   ELAT,ECONT        EPROM data latches are preset
         CLR    P0EP6+1           Write in EPROM latch to enable EPGM
         BSET   EPGM,ECONT
         BSET   VPP,PORTC         Signal that VPP can be applied
         BRA    *
*
*
*
*        The programming delay time is copied in RAM for both
*        the serial and parallel EPROM loader.
*
DELAYT   FCB    #SHORT            850 us
*
*
COUNTER  FCB    3                 counter by 4
NEWAD    FDB    0000              location for next address
*
*             lo    hi
PROG     FCB    0,0,0,0           location for prog bytes
ECHO     FCB    0,0,0,0           location for echo bytes
RX       EQU    *                 location for RX bytes,
*                                 same address as TABLE,
*                                 not used in same routines.
*
*
*        MEMORY MAP TABLE
*
*  Note : This table is used by the routine SSADD/BUMP to address
*        only relevant memory locations. It contains for each
*        EPROM segment its start address and last address +1.
*        It must be in RAM for EPROM parallel load.
*
*
TABLE    FDB    P0EP6             page 0 EPROM 6
         FDB    P0EP6+48
         FDB    EPROM1            EPROM1
```

```
EP1     FDB   EPROM1+256
        FDB   EPROM6          main EPROM 6
        FDB   EPROM6+5888
        FDB   VECT            EPROM 6 vectors
LAST    FDB   VECT+16
*
*
        FCB   $FF,$FF,$FF,$FF,$FF
*
* ==========================================================
*
* VECTORS
*
*       The unused vectors point to RAM, so as to be available
*       for test purposes (RAM Bootloader, SCI loader). Their
*       positionning allows 10 bytes for the stack, that is 2
*       interrupt levels, or 1 interrupt and 2 subroutine levels.
*
*
            ORG  BUTVCT
*
        FCB   $FF,$FF         free
*
        FDB   STACK-9-18      SCI
        FDB   STACK-9-15      TIM OVF
        FDB   STACK-9-12      TIM OUT COMP
        FDB   STACK-9-9       TIM IN CAP
        FDB   STACK-9-6       IRQ
        FDB   STACK-9-3       SWI
        FDB   ENTRY           RESET
*
* The vectors are set to $FF to avoid flashes in ROM.
*
        FCB   $FF,$FF
        FCB   $FF,$FF
        FCB   $FF,$FF
        FCB   $FF,$FF
        FCB   $FF,$FF
        FCB   $FF,$FF
        FCB   $FF,$FF
        FCB   $FF,$FF
*

              END
```

?

MC68HC705B16

Converting from the MC68HC805B6 to the MC68HC705B16

Date	Rev	Description
4/7/95	1	Original Release. Includes trackers HC705B16.006 and HC705B16.011.

Tracker Number: HC705B16.006 Revision: 2.00

BACKGROUND

During the past several years, Motorola has reviewed all its processing and manufacturing methods in search of ways to reduce the negative impact of these methods on the environment. One of the major results of this study was Motorola's decision to eliminate the use of chlorofluorocarbons (CFCs).

In compliance with this corporate policy, Motorola has discontinued the manufacture of the MC68HC805B6, whose processing used CFCs. While the recommended replacement for this device is the MC68HC705B16, it is not a drop-in replacement. The MC68HC705B16 is pin compatible and does support all of the resources found on the MC68HC805B6.

There are many considerations when converting from the EEPROM-based MC68HC805B6 to the EPROM-based MC68HC705B16. Careful attention must be paid to all of the differences to ensure successful migration to the MC68HC705B16.

In cases where the byte-programmable EEPROM is not required, or if the target MCU is the MC68HC05B4, the MC68HC705B5 provides a virtually transparent replacement for the MC68HC805B6. This EPROM-based device is also pin compatible and provides a subset of the resources found on the MC68HC805B6. The only differences are: 1) replacement of the user program memory and the byte programmable EEPROM with EPROM, 2) changing the address of the option register (OPTR) from $0100 to $1EFE, 3) the OPTR and the EEPROM control/E clock register ($0007) have been changed to support EPROM programming instead of EEPROM, and 4) the watchdog timer has been changed to include some additional options.

Many resources on the MC68HC705B16 have been implemented exactly as on the MC68HC806B6. Some resources have been changed, and several new features have been added. The following is a detailed comparison of the two devices, specifically emphasizing those subsystems which are different. In addition, any considerations which apply to converting from the MC68HC705B16 to an MROM device, such as the MC68HC05B6, will be addressed.

SYSTEM OVERVIEW

The MC68HC805B6 is an HCMOS, single-chip, 8-bit microcontroller based on the Motorola MC68HC05 CPU. The on-chip resources of this part include:

- Fully Static Operation
- On-Chip Oscillator
- Power-Saving Modes: Stop, Wait, and Slow
- 176 Bytes of Random Access Memory (RAM)
- 5952 Bytes of Electrically Erasable Programmable Read-Only Memory (EEPROM) for User Program
- 256 Bytes of Byte-Erasable EEPROM
- Internal Charge Pump
- Write/Erase Protect Bit for 224 Bytes
- 24 Bi-Directional Input/Output (I/O) Lines
- 16-Bit Free-Running Counter
- Two Input Capture Systems
- Two Output Compare Systems
- Software Force Compare
- Software Reset of Main Counter
- 8-Channel, 8-Bit Analog-to-Digital (A/D) Converter
- Two Pulse Length Modulation (PLM), 8-Bit Digital-to-Analog Converters
- Asynchronous Serial Communications Interface (SCI)
- Programmable Prescaler
- Transmitter Clock Output for Synchronous Transmissions
- External, Timer, and SCI Interrupts
- E-Clock Output Option
- Power-On Reset (POR) Bit to Determine Reset Source

The MC68HC705B16 is actually an MC68HC705X16 device which has been packaged to be compatible with the MC68HC(8)05Bx Family of MCUs. This device is normally packaged in a 64-pin PLCC or CLCC package and contains a controller area network (CAN) interface. The CAN interface is not bonded out when the MC68HC705X16 is assembled as an MC68HC705B16. The MC68HC705X16 supports all of the resources listed above, but several changes must be noted, as well as additional resources not found on the MC68HC805B6. Note that the CAN interface is not available for use in the MC68HC705B16, but must be considered in the design migration.

The MC68HC705B16 has 352 bytes of RAM, instead of the 176 bytes found in the MC68HC805B6. The 6-Kbyte EEPROM user memory space has been replaced with a

16-Kbyte EPROM array.Ports B and C have programmable pulldown resistor options. Port B also may be configured as a wired-OR interrupt source. These additional resources are certainly available for use, but if the ultimate goal is to use an MROM device, care must be taken to ensure that only those resources which will be found on the target device are used.

CPU-RELATED CONSIDERATIONS

The resources provided by the MC68HC805B6 and the MC68HC705B16 must be compared carefully to determine what changes need to be made to ensure that the system will function as desired. The first resource to be considered is the MC68HC705B16 CPU. In the MC68HC805B6, the address space is limited to an 8-Kbyte range. Only 12 internal address lines were implemented, and the upper three bits of the 16-bit program counter (PC) were always read as a zero. The address space in the MC68HC705B16 has been increased to 16 Kbytes. An additional address line was required, and the PC now has 13 active bits. The result of this change is that the MC68HC705B16 user vector table is located from $3FF2 to $3FFF. (See user vector table section.) The increased memory space also has another significant effect: The development tools used to support the MC68HC805B6 do not support development of the MC68HC705B16. (See emulation concerns section.)

Most of the differences between the MC68HC805B6 and the MC68HC705B16 are enhancements. The rest are changes required by the change from EEPROM-based user memory to EPROM-based user memory. The on-chip resources and the changes will now be discussed in a memory map sequence beginning at $0000.

PARALLEL I/O

The I/O port data registers (PORTx) and the port data direction registers (DDRx) remain at the same address locations ($0000 to $0006) and serve the same functions. But more flexibility has been added to port B and port C. These ports, when configured as inputs, now have a selectable pulldown feature. These pulldowns are selected by programming bits 0 and 1 in the mask option register (MOR) located at $3DFE. When enabled, a resistive pulldown is applied to any pins of the selected port which are configured as inputs. The pulldowns provide a sink capability of about 80 µA (62.5 KΩ), compared to the normal port input current of 1.0 µA (5 MΩ). Port B also has been enhanced with the capability to interrupt the CPU. This port can now be configured as a wired-OR interrupt source. This feature also is selected by programming bit 7 (wired-OR interrupt) in the MOR and is enabled by bit 7 (wired-OR interrupt enable) of the EPROM/EEPROM/ECLK control register at $0007.

EEPROM/E CLOCK CONTROL REGISTER

The EEPROM/ECLK control register is still located at $0007, but has been renamed the EPROM/EEPROM/ECLK control register. The bits formerly associated with control and programming of the user EEPROM space in the MC68HC805B6 now have new functions. Bit 7 was E6BW and now is WOIE. Bit 6, which was E6ERA, is now unused and is always read as zero.

A/D, PLM, SCI, AND TIMERS

The control, data, and status registers for the A/D converter, the PLM D/A converter, the SCI and the timer systems on the MC68HC705B16 are identical to those found on the MC68HC805B6 and will not require any system design changes. Also, the miscellaneous register has no changes. Addresses for these registers are $0008 to $001F.

PAGE 0 USER EEPROM 6

The page 0 user EEPROM from $0020 to $004F has been replaced with EPROM. No changes should be required in the system design.

RAM

The RAM area from $0050 to $00FF has been renamed RAM1. No changes should be required in the system design.

OPTION REGISTER

The option register (OPTR) at $0100 has been renamed EEPROM options register. No changes should required in the system design.

EEPROM1

The EEPROM1 area from $0101 to $01FF has been renamed EEPROM. There should be no changes required in the system design.

BOOTSTRAP ROM I

The bootstrap ROM I area from $0200 to $027F has been reduced to 80 bytes ($0200 to $024F). No changes should be required in the system design.

UNUSED

The UNUSED area from $0280 to $07FF has been replaced with RAM II from $0250 to $02FF. This is available for use, but, again, care must be taken to exclude this area when an MROM device is planned. No other changes should be required in the system design.

USER EEPROM 6

The EEPROM 6 area from $0800 to $1EFF has been replaced with EPROM from $0300 to $3DFD. The additional user memory space is available for use, but again care must be taken to use only valid memory space when an MROM device is planned. The system design should require no changes.

BOOTSTRAP ROM II

The bootstrap ROM II area from $1F00 to $1FEF has been moved and now is located from $3E00 to $3FEF. There should be no changes required in the system design.

USER VECTOR TABLE

The user vector table from $1FF2 to $1FFF has been moved and now is located from $3FF2 to $3FFF. One simple solution is to place the user vector table in both locations. If the goal is a smaller user memory space MROM device, the user vector table for the MC68HC705B16 should be removed in the final version of the code.

ADDITIONAL CONSIDERATIONS

The MC68HC705B16 has an additional control register, the mask option register (MOR). The MOR is used to select options which are available in the MROM devices. This register is located at $3DFE. Bit 7 (WOI) is used to control the wired-OR feature of port B. Bits 6 and 5 are not used. Bit 4 (RTIM) is used to control the reset startup time for the oscillator, either 16 or 4096 clock cycles. Bit 3 (RWAT) controls the watchdog timer's function after reset. The COP may be set to be enabled immediately out of reset or to be enabled under program control. Bit 2 (WWAT) controls the watchdog timer's function during wait mode. The timer may be disabled or enabled by this bit during wait. Bit 1 and bit 0 control the pulldown option for ports B and C.

As stated earlier, the MC68HC705B16 contains a CAN interface. A problem with the existing versions of this device occasionally allows the CAN module to be enabled after reset. While this event is rare, it is advisable to compensate for this problem by including a CAN interrupt service routine in the user software. The CAN interrupt vector is located at $3FF0-$3FF1. The suggested method is to cause a watchdog reset after a CAN interrupt occurs. The system will then restart, and the CAN module should now be disabled.

For example:

```
          ORG  $2000
CANIRQ     BSET 0,$0C      Enable Watchdog
          STOP             Cause Watchdog reset to occur
          ORG  $3FF0
          FDB   CANIRQ
```

This code stub should be placed in the upper part of the user program space, above $1FFF, making it easier to remove for migration back to an MROM device. Also, the CANIRQ vector should be removed at this time.

EMULATION CONCERNS

When migrating to the MC68HC705B16, another major issue is in development system requirements. The M68HC05EVM and the M68CDS8HC05, which currently support the MC68HC805B6, do not readily support MC68HC705B16 development. This includes programming functions as well as in-circuit emulation. It is necessary to use an M68HC05X16EVS or M68MMDS05 with an M68HC05X16EM installed to support the changes in the memory map and resource set. Cables currently used with the M68HC05EVM to support the MC68HC805B6FN can be used with the M68HC05X16EVS or the M68MMDS05. Cables used with the M68CDS8HC05 cannot be used. Neither the M68HC05X16EVS nor the M68MMDS05 supports device programming. Programming the MC68HC705B16 is only supported on the M68HC05BPGMR. The M68HC05BPGMR supports both parallel and serial programming modes. The current versions of the MC68HC705B16 require +15.5 volts for programming. Attempts to use any other programming solutions previously used with the MC68HC805B6 will be unsuccessful and may damage the devices.

EMC AND THERMAL PERFORMANCE

The technology used in the manufacture of the MC68HC805B6 is significantly different than that used in the MC68HC705B16. The device geometrics used in the MC68HC705B16 are much smaller than those used in the MC68HC806B6. The EPROM with EEPROM manufacturing process required for the MC68HC705B16 is also very different from the EEPROM-only process used for the MC68HC805B6. The MC68HC705B16 will exhibit much faster switching times and also will respond to faster input signals. System RF performance will be significantly different when using the MC68HC705B16. The user must take appropriate steps to compensate for the faster MC68HC705B16 in systems where EMC compliance is required. However, the MC68HC705B16 will actually perform more like an MROM device than the MC68HC805B6.

The die size of the MC68HC705B16 is different than that of the MC68HC805B6. The difference in die size and the change from EEPROM to EPROM will cause the thermal performance of the MC68HC705B16 to be different than that of the MC68HC805B6. There should be a net improvement in overall thermal performance with the MC68HC705B16.

FINAL COMMENTS

Applications currently using the MC68HC805B6 can take advantage of the additional resources found on the MC68HC705B16. The only drawbackcurrently is the availability of window EPROM devices, which would allow reprogramming. At this time, only one-time programmable (OTP) devices are available. The MC68HC705B16 devices should be available at the same or lower cost as the MC68HC805B6.

Applications which will migrate eventually to MROM devices can be developed successfully on the MC68HC705B16. Care must be taken to limit the resource usage to those actually found on the desired MROM device (such as MC68HC05B4, MC68HC05B6, and MC68HC05B8). Final versions of the user code must be changed to remap the user reset vectors and remove the CAN interrupt support.

Bootloader Listing

Tracker Number: HC705B16.011 Revision: 1.00

```
**********************************************************************
*
*         BOOTSTRAP LOADER for the MC68HC705B16/X16
*         =========================================
*
*         Rev   B.2                     1992/jan/14
*
*         Programmer :    E. Boulian
*
*                            -----------
*                           ( BOOTSTRAP )
*                            -----------
*                                / \
*                            1 /     \
*            +----------------< PD4 ? >
*            |                 \     /
*           / \                  \ /
*         /     \ active          | 0
*        < SEC ? >-----------------------> ( flash red led )
*         \     /                 |
*           \ /                  / \
*            | no              /     \ 1
*            |                < PD2 ? >-------+
*           / \                \     /        |
*         /     \ 1              \ /          V
*        < PD3 ? >----+         0 |   -------------------
*         \     /     |           |  ( EPROM Gate stress )
*           \ /       |           |   -------------------
*            | 0      |          / \
*            |        |        /EPROM\  no
*       -----------   |       <ERASED >-----> ( red led on )
*       |  RAM    |   |        \  ?  /
*       |  BOOT   |   |          \ /
*       |  LOADER |   |           |
*       -----------   |           |<----------+
*                     |           |           |
*            +--------+          / \          |
*            |                 /     \ 0      |
*            |                < PD1 ? >-------+
*            |                 \     /
*            |                   \ /
*            |                    | 1
*            |         ---------------------
*            |         | ERASE EPROM ARRAY |
*            |         ---------------------
*            |                    |
*            |                   / \
*            |               1 /     \
*            |<---------------< PD3 ? >
*            |                 \     /
*            |                   \ /
*            |                     | 0
*       -----------         ------------
*       |  SERIAL |         | PARALLEL |
*       | ROUTINES|         |  PROG &  |
*       |         |         | VERIFY   |
*       -----------         ------------
*
*
* NOTE : This flowchart is intended only as a mean to outline the
*        possible program paths, it is not an exact reflection of
*        all the  details of program dispatching and routing.
*
**********************************************************************
*
*
* Bootstrap loading can be done either through a parallel port
* using an external EPROM or from any other parallel source,
* using the handshake facility provided.
*
* Bootstrap loading can also be accomplished through the SCI
* in serial form, at 9600 baud for a nominal crystal frequency
* of 4 MHz. The mode selection is as follows :
*
*
*
* PORT D4 | PORT D3
* --------+--------
*   0   |   0    EPROM erase check, RAM EEPROM erase, E/EEPROM Bootstrap parallel
*    0    |    1     EPROM erase check, RAM EEPROM erase, Serial Bootstrap
*    1    |    0     Parallel RAM Bootloader
*    1    |    1     RAM/EPROM/EEPROM Serial Bootstrap
*
*
* Dump or serial load is accomplished through the same routine,
* and both functions share a common protocol, as follows :
*
*
*
* - 1) B16 sends back the 4 bytes just programmed as a prompt and also
*      for checking by the host (the first and second time this data is
*      irrelevant).
*
* - 2) B16 expects 6 bytes from HOST :
*           | ADDRESS HI:LO | DATA(a):DATA(a+1):DATA(a+2):DATA(a+3) |
*      When EEPROM1 is addressed, only DATA(a) is relevant but the 6 bytes
*      has to be sent.
*
* - 3) B16 takes a nominal 5 ms for EPROM and 10 ms for EEPROM1
*      to programm the data, while the next 6
*      bytes are being transmitted as outlined above.
*
* - 4) Loop to 1)
*
*
```

```
* Note that the program implements a pipeline structure in that 3
* operations take place simultaneously :
*
* - echo bytes are sent back to the host
* - data is being programmed
* - next data is being received
*
* It is possible to load the RAM (above address XADR)
* with a test program and execute it. Execution  is triggered
* by sending a negative (bit 7 set) high address byte. Execution
* starts at XADR.
*
*
*         I/O and INTERNAL registers definitions

*
                 ORG 0
*
PORTA     RMB    1                  port A
PORTB     RMB    1                  port B
PORTC     RMB    1                  port C
PORTD     RMB    1                  port D
DDRA      RMB    1                  port A DDR
DDRB      RMB    1                  port B DDR
DDRC      RMB    1                  port C DDR
*
ECONT     RMB    1                  EPROM/EEPROM control register
ADR       RMB    1                  A/D data register
ADC       RMB    1                  A/D status and control register
PLMA      RMB    1                  pulse length mod reg A
PLMB      RMB    1                  pulse length mod reg B
MISC      RMB    1                  miscellaneous register
BAUD      RMB    1                  SCI baud
SCCR1     RMB    1                  SCI control register 1
SCCR2     RMB    1                  SCI control register 2
SCSR      RMB    1                  SCI status register
SCDAT     RMB    1                  SCI data register
TIMC      RMB    1                  TIMER control register
TIMST     RMB    1                  TIMER status register
ICR1      RMB    2                  capture register 1 (16 BIT)
OCR1      RMB    2                  output compare register 1 (16 BIT)
TIMER     RMB    2                  TIMER free running counter (16 BIT)
DUALTM    RMB    2                  alternate counter register (16 BIT)
OCR2      RMB    2                  output capture register 2 (16 BIT)
COMP2     RMB    2                  compare register 2 (16 BIT)
*
*
```

```
*          MEMORY MAP DEFINITIONS
*
*  Note : TEST is a write only register and its access is
*         authorized only when ELAT is cleared. When ELAT
*         is set, address $20 (P0EP6) is accessed for writing.
*
*
TEST      EQU    $4F                 TEST register
P0EP6     EQU    $20                 page 0 EPROM 6 start address
RAM1      EQU    $50                 RAM1 start address
STACK     EQU    $FF                 top of STACK
EEPROM1   EQU    $100                EEPROM 1 start address
XROM      EQU    $200                Extra ROM
RAM2      EQU    $250                RAM2 start address
EPROM6    EQU    $300                EPROM 6 start address
OPTR      EQU    $3DFE               OPTION reg
BOOTR     EQU    $3E00               BOOTSTRAP ROM start address
BUTVCT    EQU    $3FE0               BOOTSTRAP ROM vectors
VECT      EQU    $3FF0               EPROM6 vectors
*
*
*          Miscellaneous definitions and equates
*
*
HI        EQU    0                   hi byte offset
LO        EQU    1                   lo byte offset
MS20      EQU    38                  20 ms timing factor
GSTR      EQU    %01100000           XCOL and XROW at 1.
LONG      EQU    $09                 long timing factor (5 ms nominal)
SHORT     EQU    $01                 short timing factor (.85 ms nominal)
LONGEE    EQU    $14                 long EEPROM1 timing factor (10 ms nominal)
SHORTEE   EQU    $03              short EEPROM1 timing factor (1.9 ms nominal)
RED       EQU    PLMA                red LED on PLMA
GREEN     EQU    PLMB                green LED on PLMB
OCF1      EQU    6                   output compare 1 flag
E1PGM     EQU    0                   ECONT  bit definition
E1LAT     EQU    1                   ECONT
E6PGM     EQU    4                   ECONT
E6LAT     EQU    5                   ECONT
SEC       EQU    0                   0 = secure
IN        EQU    6                   Handshake IN line on Port C
OUT       EQU    5                   Handshake OUT line on Port C
VPP       EQU    7                   VPP bit in portC
RDRF      EQU    5                   Receive data ready flag in SCSR
TDRE      EQU    7                   Transmit DATA Reg Empty
MBIT      EQU    4                   8 data bits flag in SCCR1
ADON      EQU    5                   A/D converter control bit
TOF       EQU    5                   Timer overflow flag
OLVL1     EQU    0                   Output level 1
OLVL2     EQU    2                   Output level 2
FOLV1     EQU    3                   Force output compare 1
FOLV2     EQU    4                   Force output compare 2
*
*
*
```

```
*
*         Synthesized instructions
*
*  Note : In order to circumvent a shortcoming inherent to any
*         2 pass assembler as the one used here, some DIRECT
*         (JSR/LDA/STA/DEC/INC) instructions have been synthesized
*         using 2 FCB instructions. This is flagged by a @@@ in
*         the comment line.
*
*
LOADA     EQU    $B6                Load A direct
STORA     EQU    $B7                Store A direct
GOTO      EQU    $BC                Jump direct
CALL      EQU    $BD                Jump to subroutine direct
DECD      EQU    $3A                Decrement direct
INCD      EQU    $3C                Increment direct
LDAX1     EQU    $E6                Load A indexed, 1 byte
STAX1     EQU    $E7                Store A indexed, 1 byte
CMPX1     EQU    $E1                Compare indexed, 1 byte
*
*
* =========================================================
*
                 ORG XROM
*
*
*  Note : Routines after the label RAMR, will be copied to
*         RAM upon initialisation. They must execute in RAM,
*         as the ROM is not available when the EPROM latch
*         bit (ELAT) is set.
*
*
*
*
* EXEC :  Gives a possibility to load and execute a program
*         in high RAM from the SCI. The main SCI loader will
*         jump here if the high address byte is ever negative.
*
*
EXEC      CLR    GREEN
          FCB    $BC                JMP instruction pointing to
          FCB    XADR-OFST          RAM, for SCI loader execution
*
*
```

```
* (this is the second part of the serial loader loop)
*
MOVE      FCB    LOADA              @@@
          FCB    RX+2-OFST          transfer RX bytes as next PROG bytes
          FCB    STORA              @@@
          FCB    PROG+2-OFST        except fourth one not received yet
          FCB    LOADA              @@@
          FCB    RX+1-OFST
          FCB    STORA              @@@
          FCB    PROG+1-OFST
          FCB    LOADA              @@@
          FCB    RX-OFST
          FCB    STORA              @@@
          FCB    PROG-OFST
          FCB    LOADA              @@@
          FCB    NEWAD+LO-OFST      transfer address
          FCB    STORA              @@@
          FCB    BPA+LO-OFST
          FCB    LOADA              @@@
          FCB    NEWAD+HI-OFST
          FCB    STORA              @@@
          FCB    BPA+HI-OFST
*
          CMP    #$01               check if EEPROM1 selected
          BEQ    EESEL
          LDA    #$01               if not set DALAYT for .850 mS
          BRA    MOV1
EESEL     LDA    #SHORTEE           if yes set DELAYT for 1.90 mS
MOV1      FCB    STORA              @@@
          FCB    DELAYT-OFST
*
          JSR    SCRD               get last data byte and put it
          FCB    STORA              @@@
          FCB    PROG+3-OFST        directly in place
*
          TST    BPA+HI-OFST        is hi address negative ?
          BMI    EXEC               if so exec
          JMP    SCILP              else go on loading
*
*
* VERFP : Verify programming, after PARPROG.
*         The content of both EPROM and EEPROM1 will be verified.
*         This routine can execute in ROM.
*
*         After PARPROG, BPS contains $C7 (STA Extended)
*
VERFP     FCB    INCD               @@@
          FCB    BPS-OFST           init BPS $C7 => $C8 (EOR)
*
          CLRX                      init address table pointer
          FCB    CALL
          FCB    SSADD-OFST         set start address
*
```

```
MOREV     BCLR   OLVL2,TIMC          clear A13 out (default)
          FCB    CALL
          FCB    XSFER-OFST          put address to ports and get data
*
          FCB    CALL                @@@
          FCB    BPS-OFST            compare [A] with EPROM or EEPROM1
          BNE    NOVERF              humm.. no good
          FCB    CALL
          FCB    BUMP-OFST           next address
          BNE    MOREV               more to go ...
*
          JMP    GREENR
*
*
NOVERF    JMP    REDR                Red LED on, green LED already off
*
*
* PROG08: Program serially in little bursts of 850 us interleaved
*         between SCI services, so as to save time.
*
PROG08    LDA     #$D7                STA IX2
          FCB     STORA               @@@
          FCB     BPS-OFST            set up BPS as STA indexed 16 bits
          JMP     PROG8RAM-OFST       continue PROG08 in RAM1
*
          FCB    $FF,$FF              two free bytes
*
* Following bytes are set to $FF to avoid flashes in ROM.
*
         FCB    $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF


         FCB    $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF


         FCB    $FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF


*
*
*====================================================================
*
                 ORG BOOTR
*
*  Note : The following routines will be executing in RAM,
*         being transfered by the BOOTSTRAP LOADER. Some data
*         is also initialized at the same time.
*
*
RAMR      EQU     *                start of routines to be copied
*                                      in RAM upon initialisation.
*
* Offset due to the moving into RAM of some XROM routines :
*
OFST      EQU    RAMR-RAM1
*
* NOTE :  The bytes affected (as far as assembly goes) by the
*         move from XROM to RAM are "corrected" by subtracting
*         OFST. This way, the generated addresses will be what
*         is required for execution in RAM. As far as reading
*         the listing is concerned, this offset can be ignored.
*
PROG8RAM  BSET    E1LAT,ECONT
          BSET    E6LAT,ECONT
          LDX     #$3               write 4 bytes every time
PLOOP     FCB     LDAX1             @@@
          FCB     PROG-OFST
          BSR     BPS
          DECX
          BPL     PLOOP
          BSR     DELAY
          RTS
*
* BPS/BPA routine; this routine, residing in RAM, is used to
*         access the entire memory space. It is modified by
*         software (sorry about that folks), and has the form :
*
*
*         | EXTENDED INSTR | 16 BIT OPERAND | RTS |
*
*
*         BPA is initialized to PROG so that on the first loop
*         of the serial loader, (dummy) programmed data does not
*         provoque an error check. It follows that the last two
*         records transmitted dummies, made up of 00 00 00 00's.
*
BPS       FCB    $C6               [LDA] 16 bit-extended
BPA       FDB    PROG              address counter/argument
          RTS                      init to non EPROM addr for SCILP
*
*
*
*
```

```
*         Delay Routine
*
*         If entered at DELAYL, the timing will be as specified by ACCA
*         and the EEPROM1 will be programmed or erased.
*         If entered at DELAY10, the timing will be 10 mS and EEPROM1
*         will be programmed.
*         If entered at DELAY, ACCA must hold a value of $01 (SHORT)
*         for a delay of 850 us.
*         If entered at DELAYX, EEPROM1 will be programmed at 10 mS
*         if X=6, EPROM is porammed at DELAYT otherwise.
*         The delay routine switches E1PGM or E6PGM on upon entry
*         except if entered at DELAYV.
*
DELAYX    CPX    #$6                 test is EEPROM selected
          BNE    DELAY
DELAY10   LDA    #LONGEE             10 mS delay
          BRA    DELAYL
DELAY     FCB    LOADA               @@@
          FCB    DELAYT-OFST         From RAM
*
DELAYL    BSET   E6PGM,ECONT         start EPROM programming
          BSET   E1PGM,ECONT         start EEPROM1 programming
          BSET   VPP,PORTC           Turn Vpp on
*
          STA    TIMER+LO            reset timer
          STA    OCR1+HI             + put OCR1H
          STA    TIMST               |    first part of clear timer status
          LDA    #$C0                timer lo byte
          STA    OCR1+LO             + put OCR1L and 2nd part of clr tim stat
*
          BRCLR  OCF1,TIMST,*        wait for preset delay
*
          BCLR   VPP,PORTC           Turn Vpp off
          CLR    ECONT               end programming
          BRCLR  7,BPA+LO-OFST,*+3  Carry reflects bit tested
          ROR    GREEN               Flash green LED at abt 3 Hz
*
          RTS
*
*
*
*  Note : None of the routines beyond this point will be
*         needed by the SCI loader. So if a program is
*         loaded in RAM, by the SCI loader, it can start
*         at this address.
*
XADR      EQU     *
*
*
*
* PARPROG: Will program all EPROM, by 8 bytes, from an
*          external program source. DATA is fetched from
*          PORT A, with or without handshake (see XSFER).
*
*          Versions before B.1 program only 4 bytes at a time.
```

```
*
PARPROG   EQU    *
*
MOREP     FCB    INCD
          FCB    COUNTER-OFST       only used if EEPROM1 selected
          CPX    #$6                check if EEPROM1 selected
          BEQ    PAR1
          LDA    #$8                if not set counter for 8 bytes
          FCB    STORA              @@@
          FCB    COUNTER-OFST
*
PAR1      BSET   E1LAT,ECONT
          BSET   E6LAT,ECONT
*
MOREL     BSR    XSFER              put address to port and get data
          BSR    BPS                store in EPROM latches
          BSR    BUMP               next address
          BEQ    LASTP
          FCB    DECD               @@@
          FCB    COUNTER-OFST
          BNE    MOREL
*
          BSR    DELAYX             go program
          BRA    MOREP              more to go ...
*
LASTP     BSR    DELAYX
          JMP    VERFP              verify, in ROM.
*
*         XSFER Routine
*
*  Note : XSFER (transfer) gets the current address from BPA,
*         puts it out to the ports, takes care of the handshake
*         protocol, and finally gets the data. If the handshake
*         is not used, connecting together IN with OUT (resp.
*         PORTA 6 & 5) will in effect disable it.
*         Note that if it is required, the OUT line (active
*         low) can be used as an Output Enable line to an
*         external EPROM.
*
*  Note : In the emulator, the end of the erase operation is
*         detected by monitoring the first falling edge of
*         the handshake OUT (PORTC5) line.
*
XSFER     FCB    LOADA              @@@
          FCB    BPA+LO-OFST        get low byte of address
          STA    PORTB              put it out
          BRCLR  5,BPA+HI-OFST,XS1  test A13
          BSET   OLVL2,TIMC         set A13 out if needed
XS1       BSET   FOLV2,TIMC
          FCB    LOADA              @@@
          FCB    BPA+HI-OFST        get high byte of address
          AND    #%00011111         set data request
          STA    PORTC              put it out
```

```
*
GETPAR    BRSET  IN,PORTC,*         wait for data valid
*
          LDA    PORTA              get data
          BSET   OUT,PORTC          acknowledge data
GOON      RTS                       [A] holds data
*
*
*         BUMP/SSADD Routine
*
*  Note : The routine must first be entered at SSADD (Set Start
*         ADDress), with X cleared. It will initialize from TABLE
*         the address for routine BPS (which accesses memory), and
*         leave the index pointing to the next table entry, that
*         is the end of the current memory segment.
*
*         When subsequently entered through BUMP, BPS address
*         (at BPA) will be incremented by one, until it is equal
*         to the current "end of memory segment" pointed to
*         by X. Then the index is incremented, to point to the
*         start address of the next segment, and control goes
*         to SSADD to initialize the address of the next segment.
*
*         Upon exit from SSADD/BUMP, the Z flag is always cleared,
*         except iff the end of table has been reached : in that
*         case only, Z is set, signaling that the whole memory
*         array has been scanned.
*
BUMP      FCB    INCD               @@@
          FCB    BPA+LO-OFST        bump lo byte
          BNE    NOC                no carry
          FCB    INCD
          FCB    BPA+HI-OFST        carry over to high byte
*
NOC       FCB    LOADA              @@@
          FCB    BPA+HI-OFST        check high byte first
          CMP    TABLE+HI-OFST2,X
          BNE    GOON               no match yet
          FCB    LOADA              @@@
          FCB    BPA+LO-OFST        check low byte then
          CMP    TABLE+LO-OFST2,X
          BNE    GOON               no match
*
          CPX    #LAST-TABLE        end of table ?
          BEQ    GOON               Z set upon exit
RETRY     INCX
          INCX                      next table entry
*
SSADD     JMP    SSAD1-OFST2
*
*
*
*
```

```
*         The programming delay time is copied in RAM for both
*         the serial and parallel EPROM loader.
*
DELAYT    FCB    #SHORT             850 us
*
*
COUNTER   FCB    3                 counter by 4
NEWAD     FDB    0000              location for next address
*
*               lo     hi
PROG      FCB    0,0,0,0           location for prog bytes
ECHO      FCB    0,0,0,0           location for echo bytes
RX        EQU    *                 location for RX bytes,
*                                  same address as TABLE,
*                                  not used in same routines.
*
* ==========================================================
*
*
RAM2R     EQU     *                start of table to be copied
*                                      in RAM2 upon initialisation.
*
* Offset due to the moving into RAM2 of some XROM routines :
*
OFST2     EQU    RAM2R-RAM2
*
*
*         MEMORY MAP TABLE
*
*  Note : This table is used by the routine SSADD/BUMP to address
*         only relevant memory locations. It contains for each
*         EPROM segment its start address and last address +1.
*         It must be in RAM for EPROM parallel load.

*
*
TABLE     FDB    P0EP6              page 0 EPROM 6
          FDB    P0EP6+48
          FDB    EEPROM1            EEPROM1
EP1       FDB    EEPROM1+256
          FDB    EPROM6             main EPROM 6
          FDB    EPROM6+15104
          FDB    VECT               EPROM 6 vectors
LAST      FDB    VECT+16
*
*
*
```

```
* GSTRAM: Sets the EPROM in Gate stress mode with data bytes
*         at $00,$00,$00,$00, and holds on with both LED's off.
*
GSTRAM    LDA    #GSTR              R/M/W not possible on $004F
          STA    TEST               OK in BOOT and NUM modes
          BSET   E6LAT,ECONT        EPROM data latches are preset
          CLR    P0EP6+1            Write in EPROM latch to enable E6PGM
          BSET   E6PGM,ECONT
          BSET   VPP,PORTC          Signal that VPP can be applied
          BRA     *
*
SSAD1     LDA    TABLE-OFST2,X      set high byte of address
          FCB    STORA              @@@
          FCB    BPA+HI-OFST
          INCX
          LDA    TABLE-OFST2,X      low byte of address
          FCB    STORA              @@@
          FCB    BPA+LO-OFST
          INCX
*
          RTS                       Z set only if end of memory
*
*
* ==========================================================

*
*
*
*         PROGRAM Section
*         ===============
*
*
*
*
*
* ENTRY POINT TO BOOTSTRAP LOADER
* ==============================
*
*
*  Note : Port A is the DATA input for MCU programming.
*         Port B and the 5 lower bits of port C reflect the
*         address being programmed. Port C, bits 5 & 6 provide
*         handshake capability while programming the MCU.
*         If the handshake is not used, ie: in case DATA
*         comes from an external EPROM, these 2 pins
*         should be tied together. Pin PORTC7 is used to switch
*         Vpp on during programming.
*         Program routing (dispatch) is accomplished by the state
*         of pins PORTD3 & PORTD4 (Check flowchart for details).
*         Led's are driven from the PLM's and will be ON when
*         the PLM's are at $00 and (mostly) OFF when they hold an
*         $FF value.
*
*
*
```

```
ENTRY     BSET   OLVL1,TIMC         Tell the world, this is a B5
          BSET   FOLV1,TIMC         by setting TCMP1
*
SETIO     LDA    #%00100000         init PORTC, hi byte of address
          STA    PORTC              handshake, and Vpp off.
          COM    RED                red led off
          COM    GREEN              green led off
*
          LDX    #RAM1              for both SCINIT & BUTRAM
*
* now let's go as fast as possible to the RAM loader
* if requested to do so.
*
          BRSET  3,PORTD,SCINIT     branch if not parallel load
*
          COM    DDRB               Port B outputs
          LDA    #%10111111         IN handshake on port C
          STA    DDRC
*
          BRCLR  4,PORTD,SCINIT     ready for RAM load and exec
*
*
* RAM BOOTSTRAP
* =============
*
*  Note : Transfer a program from an external EPROM into RAM.
*         Execution starts as soon as the RAM is full
*         (176 bytes transfered), or when PORTD4 goes low.
*
*
BUTRAM    BSR    SECBIT             check SEC bit
BR1       BRCLR  4,PORTD,EXERAM     short load
          STX    PORTB              address EPROM
          BCLR   OUT,PORTC          handshake, ready
          JSR    GETPAR             get data, with
*                                   handshake
          STA    ,X                 put to RAM 1
          INCX
          BNE    BR1                (FF => 00) = end
EXERAM    JMP    RAM1               execute in RAM 1.
*
*
*  Initialise the SCI
*
SCINIT    BCLR   MBIT,SCCR1         8 data bits
          LDA    #%11000000         baud rate 9600
          STA    BAUD
          LDA    #%00001100         TE / RE
          STA    SCCR2              end of init
          STA    SCSR               clear TDRE & TC bits
*
*  Note : Some routines, while programming, must be executing
*         from RAM, so let's start by copying them form XROM
*         to RAM. The extended addressing routine BPS must also
*         reside in RAM.
```

```
*
*
COPYL     LDA    RAMR-RAM1,X          transfer part of ROM to RAM1 and
          STA    ,X                  thus erase it. X has previously been
*                                    set to point to the bottom of RAM.
          LDA    RAM2R-RAM1,X          transfer part of TABLE to RAM2 and
          STA    RAM2-RAM1,X         thus erase it.
          INCX
          BNE    COPYL               until 176 bytes done ($FF - $50)
*
*
          BRSET  4,PORTD,SERIAL      serial load
          BRSET  2,PORTD,GSTRESS     EPROM gate stress setup
*
* Fall into ECHECK
*
* ECHECK: EPROM erase check. Read the whole array, using TABLE, and
*         check for zero.
*
ECHECK    FCB    CALL                @@@
          FCB    SSADD-OFST          Init base addr,
*                                            in RAM to save bytes
EC1       CPX    #$06                check if eeprom1
          BNE    MOREC               is addressed
          INCX                       then skip it.
          INCX
          BRA    ECHECK
MOREC     FCB    CALL                @@@
          FCB    BPS-OFST            get byte
          BNE    REDR                if not equal zero
          FCB    CALL                @@@
          FCB    BUMP-OFST           next address
          BNE    EC1
          CLRA                       all ok
          STA    GREEN               green LED on
          DECA
          STA    RED                 red LED off.
          BRCLR  1,PORTD,*           stop here if PD1 = 0
*
```

```
*
*
* VERF1 : Erasing and verify of EEPROM1. Entry is at E1ERSE.
*         If an error is detected during erase, control goes to
*         E1ERS1. There, the red led is turned on, and
*         a further attempt is made at erasing EEPROM1.
*         When EEPROM1 is erased, the red led is turned off, and
*         we go on.
*
*         BPS still contains ADD extended ($CB)
*
E1ERS1    CLR    RED                  flag erase failure
*
E1ERSE    LDA    #$80                 bulk mode  (E1BW)
          STA    TEST
ERASE     LDA    #%00000110           E1LAT+E1ERA
          STA    ECONT
          STA    EEPROM1+3            any data, address ...x11
*
          LDA    #$CC                 $CC = 104 ms (@ Xtal 4 MHz)
          FCB    CALL
          FCB    DELAYL-OFST
*         CLR    ECONT                redundant with DELAYL routine (B.1)
          CLR    TEST
*
*
VERF1     CLRX                        set base address : $100 => $1FF
VLOOP     LDA    #1
          ADD    EEPROM1,X
          BNE    E1ERS1               this byte not erased
*
          INCX                        low byte is enough
          BNE    VLOOP
          DEC    RED                  Error LED off
*
*
ERDONE    EQU    *                    End of erase
*
*
          BRSET  3,PORTD,SERIAL       do parallel if clear
*
*
```

```
*
*
*
* PARINIT:  Parallel E/EEPROM load. Init is in ROM, then jump in
*           RAM copy.
*
PARINIT   BRSET  2,PORTD,PARSH      Program time select
          LDA    #LONG              Default
          FCB    STORA              @@@
          FCB    DELAYT-OFST        In RAM
PARSH     LDA    #$C7               [STA] extended
          FCB    STORA              @@@
          FCB    BPS-OFST           set up BPS
*
*          CLRX                      init address table pointer
          FCB    CALL               @@@
          FCB    SSADD-OFST         Init base addr,
*                                           in RAM to save bytes
          FCB    GOTO               @@@
          FCB    PARPROG-OFST       program, from RAM.
** RFLASH: Flash RED LED at about 4 Hz if security bit active.
*
SECBIT    LDA    EEPROM1
          AND    #1                 check SEC bit
          BEQ    RFLASH             disallow serial loader
          RTS
RFLASH    COM    RED                toggle LED
          BRCLR  TOF,TIMST,*        Wait for timer overflow
          LDA    TIMER+LO           Clear TOF
          BRA    RFLASH             loop
*
REDR      CLRA
          STA    RED
          DECA
          STA    GREEN
          BRA    *
*
GREENR    CLRA
          STA    GREEN
          DECA
          STA    RED
          BRA    *
*
*
*
*
```

```
* GSTRESS:  Will put the EPROM in Gate stress mode for
*           reliability tests.
*
GSTRESS   JMP    GSTRAM-OFST2       Continue in RAM
*
*
*
*
* SERIAL: this is the MAIN serial (SCI) routine. It implements
*         the basic serial protocol. It is divided over both
*         BOOTROM and XROM, for ROM space usage purpose only.
*
*
SERIAL    BSR    SECBIT             check SEC bit
          BSET   7,DDRC             enable Vpp6 switch control
*
          LDA    #PROG/$100
          STA    BPA-OFST           restore initial BPA
          LDA    #PROG              destroied by ECHECK
          STA    BPA+1-OFST
*
* ECHO & PROG bytes are initialized by COPYL
* (this is the first part of the serial loader loop)
*
SCILP     FCB    LOADA              @@@
          FCB    ECHO-OFST          First echo byte
          BSR    SCWR               send it
          JSR    PROG08             first programming lap
          FCB    LOADA              @@@
          FCB    ECHO+1-OFST
          BSR    SCWR               send second echo byte
          FCB    CALL               @@@
          FCB    PROG8RAM-OFST      second programming lap
          BSR    SCRD               get new address hi byte
          FCB    STORA              @@@
          FCB    NEWAD+HI-OFST      save it
          FCB    LOADA              @@@
          FCB    ECHO+2-OFST        third echo byte
          BSR    SCWR
          FCB    CALL               @@@
          FCB    PROG8RAM-OFST      third programming lap
          BSR    SCRD               get new address lo byte
          FCB    STORA              @@@
          FCB    NEWAD+LO-OFST      save it too
          FCB    LOADA              @@@
          FCB    ECHO+3-OFST
          BSR    SCWR               last echo byte
          FCB    CALL               @@@
          FCB    PROG8RAM-OFST      fourth programming lap
          BSR    SCRD               get first (lower addressed) data byte
          FCB    STORA              @@@
          FCB    RX+0-OFST          save it
```

```
* Free CPU time in RAM
          FCB    CALL                @@@
          FCB    PROG8RAM-OFST       fifth programming lap
          BSR    SCRD                get second data byte
          FCB    STORA               @@@
          FCB    RX+1-OFST           save it
* Second free CPU time in RAM
          FCB    CALL                @@@
          FCB    PROG8RAM-OFST       sixth & last programming lap
          BSR    SCRD
          FCB    STORA               @@@
          FCB    RX+2-OFST           get & save third data byte
*REREAD    LDA    #$D6                LDA IX2
          FCB    STORA               @@@
          FCB    BPS-OFST            set up BPS as LDA indexed 16 bits
          LDX    #$3                 reread what has just been prog'd
RLOOP     FCB    CALL                @@@
          FCB    BPS-OFST
          FCB    STAX1               @@@
          FCB    ECHO-OFST           and save as next echo
*         FCB    CMPX1               @@@                \
*         FCB    PROG-OFST                              | not performed
*         BEQ    NOERR                                  | in rev. B.1
*         CLR    RED                 if error detected /
NOERR     DECX
          BPL    RLOOP
*
          JMP    MOVE
*
*
* SCRD :  Routine SCRD services the SCI, it does that by polling
*         the RDRF (received data ready flag). It returns with
*         the byte of data in ACCA.
*
*
SCRD      BRCLR  RDRF,SCSR,*         Possibly wait for char
GDATA     LDA    SCDAT               get data & clear RDRF
          RTS
*
*
* SCWR :  Routine SCWR services the SCI, it does that by polling
*         the TDRE (transmit data register empty). It sends the
*         byte of data in ACCA over the SCI link.
*
SCWR      BRCLR  TDRE,SCSR,*         Wait for previous transmission
          STA    SCDAT
          RTS
*
*
*
*
*
```

```
* VECTORS
*
*         The unused vectors point to RAM, so as to be available
*         for test purposes (RAM Bootloader, SCI loader). Their
*         positionning allows 10 bytes for the stack, that is 2
*         interrupt levels, or 1 interrupt and 2 subroutine levels.
*
*
                 ORG  BUTVCT
*
          FDB    RAM2+176-21        CAN
          FDB    RAM2+176-18        SCI
          FDB    RAM2+176-15        TIM OVF
          FDB    RAM2+176-12        TIM OUT COMP
          FDB    RAM2+176-9         TIM IN CAP
          FDB    RAM2+176-6         IRQ
          FDB    RAM2+176-3         SWI
          FDB    ENTRY              RESET
*
* The vectors are set to $FF to avoid flashes in ROM.
*
          FCB    $FF,$FF
          FCB    $FF,$FF
          FCB    $FF,$FF
          FCB    $FF,$FF
          FCB    $FF,$FF
          FCB    $FF,$FF
          FCB    $FF,$FF
          FCB    $FF,$FF
*
***************************************************
```

MC68HC705B32

Mask Set Errata

EPROM Single-Byte Programming Problem

Date	Revision	Description
4/20/95	1	Original Release. Includes tracker HC705B32.001.

Tracker Number: HC705B32.001 **Revision: 1.00**

Reference Document: MC68HC05B6/D Rev. 3, page G-5

NOTE: The information below is NOT a specification. It is NOT guaranteed. These are ESTIMATES of what the value will be once the characterization is complete and the official document is released.

Pertaining to mask set D59J:

Failure

The EPROM programming circuit fully supports the 16-byte simultaneous programming mode only and does not support single-byte programming correctly.

Description

The fault lies with the design of the EPROM array. A fault in the EPROM write data latch circuitry causes a latch to be driven to logic zero on both sides when the data bus for that bit is logic one. When the ELAT signal is removed, there is a race condition with the EPBS signal which results in the data bus value being copied to all the EPROM latches.

Work Around

Since the16-byte simultaneous programming functions correctly, it is a relatively simple matter to emulate single-byte programming by first initializing all 16 data latches to $00 and then writing the data to the appropriate address. This problem does not affect user application software in normal circumstances, since it only applies to programming the EPROM array. The serial programming software always should simulate 16-byte programming. The Motorola software for programming the 705X32 from an IBM-compatible PC is called EPBX32.EXE, written by Dugald Campbell, and functions in 16-byte programming mode. This program, therefore, programs the EPROM correctly.

A solution to this fault has been identified and shall be implemented on the next pass of silicon. Consult your local Morotola sales office.

Application explanation: In normal circumstances, this errata does not affect the user application software, but only affects software that programs the EPROM array. The parallel programming bootloader software within the 705X32 ROM performs 16-byte programming and so functions correctly.

Bootloader for Mask Set D59J

Tracker Number: HC705B32.002   Revision: 1.00

Mask Set: D59J

```
********************************************************************
*
*       BOOTSTRAP LOADER for the MC68HC705B32/X32
*       ========================================
*
*       Rev   1.0       15 Jan. 1993
*
*       Rev   1.1       25 Jan. 1993
*
* - check SEC before entry to VERF (verify routine) only
* - during byte erase, toggle pin only if SEC non-active
*
*       Programmer :   A. Dobbin
*
*       Description : B32/X32 bootloader completely re-writen and functions
*                     modified from earlier B16/X16 version.  Parallel
*                     E/EEPROM programming h/w compatible w/ earlier versions.
*
*
* The 705X32 bootloader differs from the 705X16 as follows :
*
* - Gate stress routine removed.
* - Parallel RAM load removed.
* - Serial RAM load/execute data format altered.  Now expects count
*   byte, followed by data bytes then jumps to $0051 once 'count'
*   bytes received.  If count is larger than 176 then code is also
*   loaded into RAM2 then RAM3 until full or until a 5ms idle delay is
*   detected.
* - Geneva TALKER05 code added - 4 bytes received serially on PA0.
* - Jump to RAM at $0051 added.
* - Parallel program/verify EPROM only added.
* - Parallel EPROM only or E/EEPROM verify added.
* - EEPROM erase routine changed from bulk erase to byte erase -
*   PLMA toggles with each byte erased.
* - Parallel programming routine 'skips' any EEPROM bytes or EPROM
*  16 bytes that are in the erased state to speed up programming
* routine.
* - Vectors now pointing to start of RAM1 area to keep SWI position
*   compatible with TALKER05 use.
*
**********************************************************************
*
*       PIN DESCRIPTION
*       ---------------
*
* Serial modes
*--------------
* PA0          : Receive/transmit pin for TALKER
* RDI          : Receive pin for RAM load/execute
*
* Parallel modes
*----------------
* Port A       : DATA input
* Port B       : Low address byte A0-A7
* PC0-PC4      : Upper address byte A8-A12
* TCMP2        : A13
* TCMP1        : A14
* PC5,6        : Data handshake. Link if external EPROM used.
* PC7          : Switch on Vpp during programming.
* PLMA, PLMB   : Red, Green LEDs. (ON when PLM = 00).
* PD0,5,6,7    : Not used.
* PD1,2,3,4    : Mode selection as follows :
```

```
*
*-----------------------------------------------------------------------------
* PD4  PD3  PD2  PD1    MODE
*-----------------------------------------------------------------------------
*  0    0    0    0     E erase check
*  0    0    0    1     E erase check, erase EE, parallel E/EE 5ms prog/verify
*  0    0    1    0     Parallel E only verify (SEC non active)
*  0    0    1    1     E erase check, erase EE, parallel E only 5ms prog/verify
*-----------------------------------------------------------------------------
*  0    1    0    0     TALKER05 (SEC non active)
*  0    1    0    1            "
*  0    1    1    0     Jump to RAM $0051 (SEC non active)
*  0    1    1    1            "
*-----------------------------------------------------------------------------
*  1    0    0    0     E erase check
*  1    0    0    1     E erase check, erase EE, parallel E/EE 0.9ms prog/verify
*  1    0    1    0     Parallel E and EE verify (SEC non active)
*  1    0    1    1     E erase check, erase EE, parallel E only 0.9ms prog/verf
*-----------------------------------------------------------------------------
*  1    1    X    X     Serial RAM load/execute (SEC non active).
*-----------------------------------------------------------------------------
*
*       X = don't care



*        I/O and INTERNAL registers definitions
*
                ORG 0
*
PORTA   EQU     00      port A
PORTB   EQU     01      port B
PORTC   EQU     02      port C
PORTD   EQU     03      port D
DDRA    EQU     04      port A DDR
DDRB    EQU     05      port B DDR
DDRC    EQU     06      port C DDR
*
```

```
ECONT   EQU     07      EPROM/EEPROM control register
ADR     EQU     08      A/D data register
ADC     EQU     09      A/D status and control register
PLMA    EQU     $0A     pulse length mod reg A
PLMB    EQU     $0B     pulse length mod reg B
MISC    EQU     $0C     miscellaneous register
BAUD    EQU     $0D     SCI baud
SCCR1   EQU     $0E     SCI control register 1
SCCR2   EQU     $0F     SCI control register 2
SCSR    EQU     $10     SCI status register
SCDAT   EQU     $11     SCI data register
TIMC    EQU     $12     TIMER control register
TIMST   EQU     $13     TIMER status register
ICR1    EQU     $14     capture register 1 (16 BIT)
OCR1    EQU     $16     output compare register 1 (16 BIT)
TIMER   EQU     $18     TIMER free running counter (16 BIT)
DUALTM  EQU     $1A     alternate counter register (16 BIT)
OCR2    EQU     $1C     output capture register 2 (16 BIT)
COMP2   EQU     $1E     compare register 2 (16 BIT)
*
*
*       MEMORY MAP DEFINITIONS
*
*
TEST    EQU     $4F     TEST register
RAM1    EQU     $50     RAM1 start address
RAMST   EQU     RAM1
STACK   EQU     $FF     top of STACK
EEPROM1 EQU     $100    EEPROM 1 start address
BOOTR1  EQU     $200    BOOTROM 1
RAM2    EQU     $250    RAM2 start address
RAM3    EQU     $300    RAM3 start address
BOOTR2  EQU     $3B0    BOOTROM 2
EPROM6  EQU     $400    EPROM 6 start address
BOOTR3  EQU     $7E00   BOOTROM 3
MOR     EQU     $7FDE   Mask option register
BOOTV   EQU     $7FE0   BOOTSTRAP ROM vectors
```

```
VECT    EQU     $7FF0   EPROM6 vectors
*
*
* RAM DEFINITIONS
*
        ORG     RAMST
*
SCRAT   RMB     4              4 contiguous bytes in ram
*
*
*       Miscellaneous definitions and equates
*
RXB       EQU    0              Receive bit for TALKER
HI        EQU    0              hi byte offset
LO        EQU    1              lo byte offset
LONG      EQU    $09            long timing factor (5 ms nominal)
SHORT     EQU    $01            short timing factor (0.90 ms nominal)
LONGEE    EQU    $13            long EEPROM1 timing factor (10 ms nominal)
RED       EQU    PLMA           red LED on PLMA
GREEN     EQU    PLMB           green LED on PLMB
OCF1      EQU    6              output compare 1 flag
E1PGM     EQU    0              ECONT  bit definition
E1LAT     EQU    1              ECONT
E6PGM     EQU    4              ECONT
E6LAT     EQU    5              ECONT
SEC       EQU    0              0 = secure
IN        EQU    6              Handshake IN line on Port C
OUT       EQU    5              Handshake OUT line on Port C
VPP       EQU    7              VPP bit in portC
RDRF      EQU    5              Receive data ready flag in SCSR
TDRE      EQU    7              Transmit DATA Reg Empty
MBIT      EQU    4              8 data bits flag in SCCR1
ADON      EQU    5              A/D converter control bit
TOF       EQU    5              Timer overflow flag
OLVL1     EQU    0              Output level 1
OLVL2     EQU    2              Output level 2
FOLV1     EQU    3              Force output compare 1
```

```
FOLV2     EQU    4                 Force output compare 2
*
*
******************
* START OF CODE
******************
*
       ORG     BOOTR3
*
* THIS CODE WILL BE COPIED FROM BOOTROM3 TO RAM1
*
RAM1R   EQU     *              Start of code to run from RAM1
OFST1   EQU     RAM1R-RAM1      Ofset for addressing this block of code.
*
* These are variables whose initial values will be transfered into RAM.
*
DELAYT  FCB     #SHORT          850 us
COUNTER FCB     00              Counter
BLANK   FCB     00              Set to 00 if programming data is blank
                                (erased state)
SECOPY  FCB     00              Copy of EEPROM security byte

BPS     FCB     $C6             [LDA] 16 bit extended
BPA     FDB     0000            16 bit address
       RTS

*        MEMORY MAP TABLE
*
*  Note : This table is used by the routine SSADD/BUMP to address
*        only relevant memory locations. It contains for each
*        E/EEPROM segment its start address and last address +1.
*        It must be in RAM for EPROM parallel load.
*
TABLE   FDB     EEPROM1         Start of EEPROM1
       FDB     EEPROM1+256     End of EEPROM1+1
       FDB     EPROM6          Start address of EPROM
       FDB     BOOTR3          End address of EPROM+1
```

```
        FDB     VECT            Start address of EPROM vectors
        FDB     VECT+16           End address of EPROM vectors+1
        FDB     MOR             Mask option register programmed last so that
LAST    FDB     MOR+1           exit from PARPROG loop possible once single
*                               byte latched.
*
*
* THIS CODE WILL BE COPIED FROM BOOTROM3 TO RAM2
*
RAM2R   EQU     *               Start of code to run from RAM2
OFST2   EQU     RAM2R-RAM2      Ofset for addressing this block of code.
*
*
*       BUMP/SSADD Routine
*
*  Note : The routine must first be entered at SSADD (Set Start
*       ADDress), with X cleared. It will initialize from TABLE
*       the address for routine BPS (which accesses memory), and
*       leave the index pointing to the next table entry, that
*       is the end of the current memory segment.
*
*       When subsequently entered through BUMP, BPS address
*       (at BPA) will be incremented by one, until it is equal
*       to the current "end of memory segment" pointed to
*       by X. Then the index is incremented, to point to the
*       start address of the next segment, and control goes
*       to SSADD to initialize the address of the next segment.
*
*       Upon exit from SSADD/BUMP, the Z flag is always cleared,
*       except iff the end of table has been reached : in that
*       case only, Z is set, signaling that the whole memory
*       array has been scanned.
*
BUMP    INC     <BPA+LO-OFST1           Bump lo address byte
        BNE     NOC                     No carry
        INC     <BPA+HI-OFST1           Carry over to high byte
*
```

```
NOC     LDA     <BPA+HI-OFST1           Check high byte first
        CMP     <TABLE+HI-OFST1,X
        BNE     GOON                    No match yet
        LDA     <BPA+LO-OFST1           Check low byte then
        CMP     <TABLE+LO-OFST1,X
        BNE     GOON                    No match
*
        CPX     #LAST-TABLE             End of table ?
        BEQ     GOON                    Z set upon exit
RETRY   INCX
        INCX                            Next table entry
*
* fall into SSADD
*
* This routine copies an E/EEPROM block starting address from TABLE into the
* RAM subroutine address (BPA). Block selected depends on X.
****************************************************************************
SSADD   LDA     <TABLE-OFST1,X          Get high byte of address
        STA     <BPA+HI-OFST1           Store in RAM sub.
        INCX
        LDA     <TABLE-OFST1,X          Get low byte of address
        STA     <BPA+LO-OFST1           Store in RAM sub.
        INCX
*
GOON    RTS                             Z set only if end of memory
*
*
*       Delay Routine
*
*       If entered at DELAYL, the timing will be as specified by ACCA
*       and the EEPROM1 will be programmed or erased.
*       If entered at DELAY10, the timing will be 10 mS and EEPROM1
*       will be programmed.
*       If entered at DELAY, ACCA must hold a value of $01 (SHORT)
*       for a delay of 850 us.
*       If entered at DELAYX, EEPROM1 will be programmed at 10 mS
*       if X=6, EPROM is prorammed at DELAYT otherwise.
*       The delay routine switches E1PGM or E6PGM on upon entry
*       except if entered at DELAYV.
*
DELAYX  CPX     #$2             Test if EEPROM selected
        BNE     DELAY
DELAY10 LDA     #LONGEE         10 mS delay
        BRA     DELAYL
DELAY   LDA     <DELAYT-OFST1   From RAM
*
DELAYL  BSET    E6PGM,ECONT     Start EPROM programming
        BSET    E1PGM,ECONT     Start EEPROM1 programming
        BSET    VPP,PORTC       Turn Vpp on
*
        STA     TIMER+LO        Reset timer
        STA     OCR1+HI         Store delay value in OCR1 hi byte
        STA     TIMST           First part of clear timer status
        LDA     #$C0            Timer lo byte
        STA     OCR1+LO         Store low delay byte and 2nd part of clr tim
*                               stat
*
        BRCLR   OCF1,TIMST,*    Wait for preset delay
*
        BCLR    VPP,PORTC               Turn Vpp off
        CLR     ECONT                   End programming
        BRCLR   0,BPA+HI-OFST1,FLSHG    Carry reflects bit tested
FLSHG   ROR     GREEN                   Flash green LED at abt 6 Hz
*
        RTS
*
*
*
* PARPROG: Will program all EPROM by 16 bytes, EEPROM by 1 byte from an
*       external program source. DATA is fetched from PORT A, with or
*       without handshake (see XSFER). If data is in erased state then
*       no programming is done on that byte/16 bytes. COUNTER must be 00
*       at entry.
*
```

```
PARPROG   EQU    *
*
MOREP   INC     <COUNTER-OFST1  Only used if EEPROM1 selected
        CLR     <BLANK-OFST1    Clear 'blank data' register initially.
        CPX     #$2             Check if EEPROM1 selected
        BEQ     PAR1
        LDA     #$10            If not set counter for 16 bytes
        STA     <COUNTER-OFST1
*
PAR1    BSET    E1LAT,ECONT
        BSET    E6LAT,ECONT
*
MOREL   BSR     XSFER           Put address to port and get data
        JSR     <BPS-OFST1      Store data in E/EEPROM latches
*
        CPX     #$2             Check if EEPROM1 selected.
        BNE     NOTEE
        COMA                    Complement to give 00 if EEPROM data
        BRA     DONE              = $FF (erased state).
NOTEE   ORA     <BLANK-OFST1    OR all 16 bytes EPROM data to give 00 only if
*                               all $00(erased state)
DONE    STA     <BLANK-OFST1    Store value. If = 00, then this data needn't be
*                               programmed.
*
        BSR     BUMP            Next address
        BEQ     LASTP           Z set once end of memory map reached.
        DEC     <COUNTER-OFST1
        BNE     MOREL           Repeat for 16 bytes.
*
        LDA     <BLANK-OFST1    Skip programming step if 16 bytes EPROM/1 byte
        BEQ     NOPROG            EEPROM data to be programmed is in erased state.
        BSR     DELAYX          Else go program
NOPROG  CLR     ECONT           Reset latch bits to clear data latches.
        BRA     MOREP           More to go ...
*
LASTP   BSR     DELAYX
        JMP     VERFP           Verify, in ROM.
```

```
*
*       XSFER Routine
*
*  Note : XSFER (transfer) gets the current address from BPA,
*       puts it out to the ports, takes care of the handshake
*       protocol, and finally gets the data. If the handshake
*       is not used, connecting together IN with OUT (resp.
*       PORTA 6 & 5) will in effect disable it.
*       Note that if it is required, the OUT line (active
*       low) can be used as an Output Enable line to an
*       external EPROM.
*
*  Note : In the emulator, the end of the erase operation is
*       detected by monitoring the first falling edge of
*       the handshake OUT (PORTC5) line.
*
XSFER   LDA     <BPA+LO-OFST1          Get low byte of address
        STA     PORTB                  Put it out
        BCLR    OLVL2,TIMC             Clear A13
        BRCLR   5,BPA+HI-OFST1,XS1     Test A13
        BSET    OLVL2,TIMC             Set A13 if reqd.
XS1     BSET    FOLV2,TIMC             Force output on TCMP2
        BCLR    OLVL1,TIMC             Clear A14
        BRCLR   6,BPA+HI-OFST1,XS2     Test A14
        BSET    OLVL1,TIMC             Set A14 if reqd.
XS2     BSET    FOLV1,TIMC             Force output on TCMP1
        LDA     <BPA+HI-OFST1          Get high byte of address
        AND     #%00011111             Set data request
        STA     PORTC                  Put it out
*
        BRSET   IN,PORTC,*             Wait for data valid
*
        LDA     PORTA                  Get data
        BSET    OUT,PORTC              Acknowledge data
        RTS     [A] holds data
*
*
```

```
*
*******************************************************************
* END OF RAM ROUTINES
*******************************************************************
*
*
*
* ENTRY POINT TO BOOTSTRAP LOADER
* ==============================
*
START   EQU     *
*
        SEI
        RSP
        BRCLR   3,PORTD,PARAL   Parallel modes
        JSR     SECBIT          Check security before entering serial routines.
        JMP     SERIAL          Enter serial routines.
*
* Initialise for parallel routines
*
PARAL   LDA     #%00100000      Init PORTC, hi byte of address
        STA     PORTC           Handshake, and Vpp off.
        COM     RED             Red led off
        COM     GREEN           Green led off
*
        COM     DDRB            Port B outputs
        LDA     #%10111111      IN handshake on port C
        STA     DDRC
*
*
*  Note : Some routines must run from RAM since BOOTROM disappears while
*         programming EPROM.  This routine copies code from the start of
*         BOOTROM3 into RAM2.
*
*
        CLRX
COPYR1  LDA     RAM1R,X         Read code from ROM ....
```

```
        STA     RAM1,X          .... and store it in RAM1.
        INCX
        CPX     #RAM2R-RAM1R    Repeat until start of RAM2R block:
*                               ROM -> $50-$63
        BNE     COPYR1

        CLRX
COPYR2  LDA     RAM2R,X         Read code from ROM ....
        STA     RAM2,X          .... and store it in RAM2.
        INCX
        BNE     COPYR2          Repeat until 256 bytes done: ROM -> $250-$34F
* note - not all 256 bytes required but done anyway.
*
        LDA     EEPROM1
        STA     <SECOPY-OFST1   Save contents of SECURITY byte
*
        BRCLR   2,PORTD,ECHECK
        BRSET   1,PORTD,ECHECK
        JSR     SECBIT          Check security non-active before entry to
*                               verify routines.
        BRCLR   4,PORTD,VERFP   Verify EPROM only programmed correctly
        BRA     VERFP2          Verify E/EEPROM programmed correctly (X=0)
*
* ECHECK: EPROM erase check. Read the whole array, using TABLE, and
*         check for zero.
*
ECHECK  LDX     #04
        JSR     SSADD-OFST2     Init base addr in RAM to start of EPROM
EC1     JSR     <BPS-OFST1      Get byte
        BNE     REDR            if not equal zero
        JSR     BUMP-OFST2      Next address
        BNE     EC1
        CLRA                    All ok
        STA     GREEN           Green LED on
        DECA
        STA     RED             Red LED off.
*
```

```
        BRCLR   1,PORTD,*       Stop here if PD1 = 0
*
*
* E1ERSE : Byte erase EEPROM1 and verify byte erased.  Repeated this
*          routine till all bytes erased.  Red LED toggles with every byte
*          erased (only if SEC non-active).  This routine is used to clear
*          the security byte for access to different modes. The security byte
*          must be erased last.
*
* NOTE: X16 code used BULK erase. Changed here to BYTE erase since no external
*       Vpp1 is connected on programming boards.
*
*
E1ERSE  CLR     <BPA+HI-OFST1   This ensures Green LED stays on during DELAY
*                               routine
        CLRX                    Set pointer to last byte in array+1
E1ERS1  DECX                    Check next byte
E1ERS2  LDA     #%00000110      E1LAT+E1ERA
        STA     ECONT
        STA     EEPROM1,X       Latch any data
        JSR     DELAY10-OFST2   Byte erase for 10ms
* Now verify byte erased
        LDA     EEPROM1,X
        INCA
        BNE     E1ERS2          Repeat if byte not erased
*
        BRCLR   0,<SECOPY-OFST1,NOFLSH  No toggle if SEC bit is 0.
        COM     RED             Toggle pin to show array condition
NOFLSH  CPX     #00
        BNE     E1ERS1          Repeat until SEC erased.
*
        LDA     #$FF
        STA     RED             Error LED off
*
* End of EEPROM1 erase
*
*
```

```
* PARINIT:  Parallel E/EEPROM load. Init is in ROM, then jump in
*           RAM copy. X = 00 at entry.
*
PARINIT BRSET   4,PORTD,PARSH   Program time select
        LDA     #LONG           Default
        STA     <DELAYT-OFST1   In RAM
PARSH   LDA     #$C7            [STA] extended
        STA     <BPS-OFST1      Set up BPS
*
        BRCLR   2,PORTD,INITBA  Default to EEPROM base address
        LDX     #$4             Initialise base address for EPROM only
INITBA  JSR     SSADD-OFST2     Init base addr,
*                               in RAM to save bytes
        JMP     PARPROG-OFST2   Program, from RAM.

* VERFP : Verify programming, after PARPROG.
*         The content of both EPROM and EEPROM1 will be verified.
*         This routine can execute in ROM.
*
VERFP   CLRX                    Init address table pointer
        BRCLR   2,PORTD,VERFP2  Default to EEPROM base address
        LDX     #$4             Initialise base address for EPROM only
VERFP2  JSR     SSADD-OFST2     Set start address
        LDA     #$C8
        STA     <BPS-OFST1      $C8 (EOR)  -> BPS
*
MOREV   JSR     XSFER-OFST2     Put address to ports and get data
*
        JSR     <BPS-OFST1      Compare [A] with EPROM or EEPROM1
        BNE     REDR            humm.. no good
        JSR     BUMP-OFST2      Next address
        BNE     MOREV           More to go ...
*
* Verify complete
*
GREENR  CLRA
        STA     GREEN           Green on
        DECA
        STA     RED             Red off
        BRA     *
*
REDR    CLRA
        STA     RED             Red on
        DECA
        STA     GREEN           Green off
        BRA     *
*
**********************************************************************
* SUBROUTINES
**********************************************************************
*
SECBIT  LDA     EEPROM1
        AND     #1              Check SEC bit
        BEQ     RFLASH          Hang-up if security bit is 0.
        RTS                     Return if security non-active
*
RFLASH  COM     RED             Flash RED LED at about 4 Hz if security bit
*                               active.
        BRCLR  TOF,TIMST,*      Wait for timer overflow
        LDA    TIMER+LO         Clear TOF
        BRA    RFLASH           Loop
*
********************************************************************************
* SERIAL ROUTINES ENTRY POINT
********************************************************************************
*
*

SERIAL  BRSET   4,PORTD,LDRAM   Do serial ram load/execute
        BRSET   2,PORTD,JMPRAM  Jump to RAM at $0051
        JMP     TALK            Do Geneva TALKER05 routine.
*
********************************************************************************
********************************************************************************
```

```
*
*     SERIAL LOAD RAM AND EXECUTE
*
* The data should be in the form of 1 start-bit, 8 data-bit, 1 stop-bit.
* Baud rate is set to 9600 baud (using a 4MHz crystal).
* 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 address $0050,
* so the first program byte will be loaded into address $0051.
* That is where program execution will begin.
*
* If the count byte is larger than the size of RAM1, ie, above 176 then
* the code continues to fill RAM2 then RAM3. In this case the count byte is
* ignored and program execution begins once all RAM is filled or if no
* character is received for 5 milli secs.  Program execution still begins
* at $0051.
*
* The user must take care when using branches or jumps in their source code!!!
* Since the program overwrites the stack area, the user should send NOPs to
* temporary 'fill up' their required stack area.
*
* This routine does't use stack or RAM so all bytes can be filled.
*
*******************************************************************************
LDRAM   EQU     *
*
        LDA     #$C0
        STA     BAUD            Set baud rate to 9600
        CLR     SCCR1           8 data bits
        BSET    2,SCCR2         Enable SCI receiver

        LDX     #RAMST          Point to start of RAM1
WAITRX  BRCLR   RDRF,SCSR,*     Wait for receive register to fill
        LDA     SCDAT           Read data byte
        STA     ,X              Store the data in RAM
        DEC     RAMST           Decrement the count byte (1st byte sent)
        BEQ     JMPRAM          Start executing code in RAM once zero.
        INCX                    Move to next RAM location
        BNE     WAITRX          Continue loading until RAM1 full

* Count byte was bigger than 176, so start filling RAM2 then RAM3 until
* space of 5 millisecs or until RAM3 filled.

        LDX     #RAMST          Point to start of RAM2
FILLR2  CLR     OCR1            Use OCR1 as a variable
        LDA     #3              Set for 5msec delay
WAITR2  BRSET   RDRF,SCSR,OUT1  [5] Wait for receive register to fill
        DEC     OCR1            [5]
        BNE     WAITR2          [3]
        DECA
        BNE     WAITR2
        BRA     JMPRAM          Execute code after timeout occurs
*
OUT1    LDA     SCDAT           Read data byte
        STA     $0200,X         Store the data in RAM2
        INCX                    Move to next RAM location
        BNE     FILLR2          Continue loading until RAM2 full

* RAM2 filled, start to fill RAM3.

        CLRX                    Point to start of RAM3
FILLR3  CLR     OCR1            Use OCR1 as a variable
        LDA     #3              Set for 5msec delay
WAITR3  BRSET   RDRF,SCSR,OUT2  [5] Wait for receive register to fill
        DEC     OCR1            [5]
        BNE     WAITR3          [3]
        DECA
        BNE     WAITR3
        BRA     JMPRAM          Execute code after timeout occurs
*
OUT2    LDA     SCDAT           Read data byte
        STA     $0300,X         Store the data in RAM3
        INCX                    Move to next RAM location
        CPX     #$B0
        BNE     FILLR3          Continue loading until RAM3 full
```

```
* fall into JMPRAM once RAM3 filled.

JMPRAM  JMP    RAMST+1        Start executing code in RAM

*
******************************************************************************
*
*    TALKER05 RAM MONITOR FIRMWARE
*
*    RAM+ROM size optimised test communication concept for HC05 under test.
*    The protocol consists of downloading serially 4 bytes and expecting 1
*    byte back as an answer to a command. The first 3 bytes are placed in
*    RAM at location 'SCRAT', the 4th byte is placed into Acc, then SCRAT+3
*    is written with $81 (RTS). The processor executes the code placed in
*    SCRAT as a subroutine and then sends back the accumulator content. Jump
*    command can be associated with an acc value for particular test purpose.
*    Serial transfer is achieved via port 'PORTTK,RXB' using a software
*    serial async routine. Testing of the SCI is possible independently.
*
******************************************************************************
*    Version for 9600 baud @ 4MHz,    Bit waste time = fosc / 2 / 104 cycles
******************************************************************************

        ORG    BOOTR2

TALK    EQU    *              TALKER05 STARTING POINT
TLOOP   BCLR   RXB,DDRA       port as input
        LDX    #SCRAT         point to scratch

**** Get 4 bytes from host ****

GBYTE   BRSET  RXB,PORTA,*    [5]- 5  wait for start bit
        LDA    #50            [2]  7  start bit: length 1.5 bit
LP1     DECA                  [3]     waste time
        BNE    LP1            [3]307
        LDA    #$80           [2]309  prepare memory for receive
        STA     ,X              [3]312

GBIT    BRCLR   RXB,PORTA,GB2   [5]- 5  get bit into carry
GB2     LDA     #32             [2]  7  waste time
LP2     DECA                    [3]
        BNE     LP2             [3]199
        ROR     ,X              [5]204  shift it

        BCC     GBIT            [3]209  not yet 8th bit
        BRCLR   RXB,PORTA,TLOOP wrong stop bit: frame error = reset
        LDA     ,X              4th byte in acc
        INX
        CPX     #RAMST+4        4th byte?
        BNE     GBYTE           get next byte

        LDX     #$81            opcode RTS
        STX     SCRAT+3         into 4th address
LP3     INX                     waste time till stop bit ended
        BNE     LP3             760 cycles = 380 uS

**** Jump into downloaded routine: ****
* A=last value txed, X=0, Z=1, C=0, N=1

        JSR     SCRAT           execute now

**** Return 1 byte to host ****

        SEC                     prepare stop bit
        BSET    RXB,DDRA        port as output
**                          .   start bit: length 1.0 bit
STBIT   BCLR    RXB,PORTA       [5]- 5 - 5      is a "0"

OBIT    LDX     #32             [2]  7  10      waste time
LP4     DECX                    [3]
        BNE     LP4             [3]199 202
        RORA                    [3]203 205      shift
        BCC     STBIT           [3]206 208      check bit value
        CLC                 [2]208 210     timing -2 to +2 per bit
        BSET   RXB,PORTA      [5]-   - 5     is a "1"
        BNE    OBIT           [3]     8     acc not equal 0

        BRA    TLOOP          loop forever

*
*
*******************************************************************
*
* BOOTLOADER VECTORS (RESET VECTOR MUST BE AT $7FEE AND 7FEF)
*
*
* VECTORS
*
* The unused vectors point to RAM1, so as to be available for test
* purposes (RAM Bootloader, SCI loader). Their positioning at start
* of RAM1 is consistent with TALKER PC program which assumes SWI vector
* is at RAMST+4.
*
        ORG    BOOTV

        FDB    RAMST+22     CAN
        FDB    RAMST+19     SCI
        FDB    RAMST+16     TIM OVF
        FDB    RAMST+13     TIM OUT COMP
        FDB    RAMST+10     TIM IN CAP
        FDB    RAMST+7      IRQ INTERRUPT VECTOR
        FDB    RAMST+4      SOFTWARE INTERRUPT (PLACE HERE TO SUIT TALKER)
        FDB    START        RESET VECTOR
*
        END

?
```