AN434

Serial bootstrap for the RAM and EEPROM1 of the MC68HC05B6

By Jeff Wright,
Motorola Ltd., East Kilbride

INTRODUCTION

The MC68HC05B6 has 256 bytes of on chip EEPROM, called EEPROM1, which can be used to store variable data in a non-volatile manner. In many applications this EEPROM1 will be used to hold a look-up table or system set up variables. In these cases it is usually a requirement that the EEPROM1 be initialised during the manufacture of the application. In addition, loading small programs into RAM and executing them is an easy way of trying out new software routines. This application note describes one method for serially loading (bootstrapping) the EEPROM1 via a program executing in the RAM of the MC68HC05B6.

BUILT IN BOOTSTRAP

The MC68HC05B6 has a built in RAM serial bootstrap program contained in the mask ROM of the device that uses the SCI. It would therefore seem a simple task to load programs into RAM; however, as ROM space on the device is obviously critical, a very simple protocol has been implemented. This means that the bootloader on the 'B6' does not accept S-records which are the normal output from an assembler; instead, the protocol expects pure binary data preceded by a count byte that holds the size of the program to be downloaded. No address information is contained in the download; instead, the bootloader always starts the program load at address $50 in RAM. The first byte (the count byte) is stored here and then as the subsequent bytes are received via the SCI they are stored at incrementing RAM locations and the count byte is decremented for each byte received. When the count byte reaches zero the bootstrap program jumps to address $51 and starts to execute the program that has just been loaded. No built in bootstrap routine is provided for the EEPROM1 array.

These restrictions present two problems:

i) How to convert assembler output to the format accepted by the 68HC05B6 RAM bootstrap routine?

ii) How to bootstrap the EEPROM1 of the 68HC05B6?

This application note provides a solution for each of these problems.

1) CONVERTING S-RECORDS FOR RAM BOOTSTRAP

To use the built in RAM bootstrap program on the MC68HC05B6 the device must be configured as shown in Figure 1. If these conditions are met when the reset pin is released, then the serial bootstrap program described above will start to execute and a program can be downloaded via a 9600 baud RS-232 source. Personal computers usually have one or more RS-232 ports referred to as COM ports. To overcome the format difference between S-records and that accepted by the bootloader, a conversion program is required. There is also an additional problem when using a PC – when a file is copied to a COM port to transfer it, it is the ascii characters that are transmitted, not the binary data. This means for example that if a file containing the typed data byte $A5 was copied via the COM port to the B6, the B6 would in fact receive two bytes: $41 and $35, which represent the ascii characters A and 5 respectively.

This means that the conversion program has to strip out the S-record format and convert the resultant data to binary format for transfer to the HC05B6. It must also insert the count byte at the beginning of the output file.

The pascal program BINCONV performs these three tasks; a listing of the source code is given at the end of this application note. A flow diagram of BINCONV can be seen in Figure 2. The inclusion of the count byte has been left as an option to increase the flexibility of the program, but it could easily be standardised to include the count byte for the B6 RAM bootloader. When BINCONV is invoked it prompts for the name of the S-record input file and the name required for the binary output file. After this each S-record in the input file is read and converted to binary data and stored in a temporary file. As each S-record is read it is echoed to the screen; when they have all been processed a message prompts the user and asks if a count byte is required. When used with the 68HC05B6 RAM bootloader the answer will always be yes, in which case the count value is written to the output file before the rest of the data is copied from the temporary file to the output file. Finally the value of the count byte is displayed for user confirmation – remember that the count byte is equal to the number of bytes in the program being converted plus 1 for the count byte itself. The program will only accept standard S-record format and will trap and abort if any non-valid character or format is detected.

With the PC COM port set for 9600 baud and the 68HC05B6 configured as in Figure 1 the binary file can be transferred and executed as follows:

i) Release Reset on the HC05B6

ii) Enter the command "COPY XXXX.YYY COM1\B" on the PC.

The program will then be transferred to the B6 and execution started automatically. Note that the \B option is used to denote a binary file transfer so that the copy procedure does not abort if it finds an end of file (EOF) character in the middle of the file.

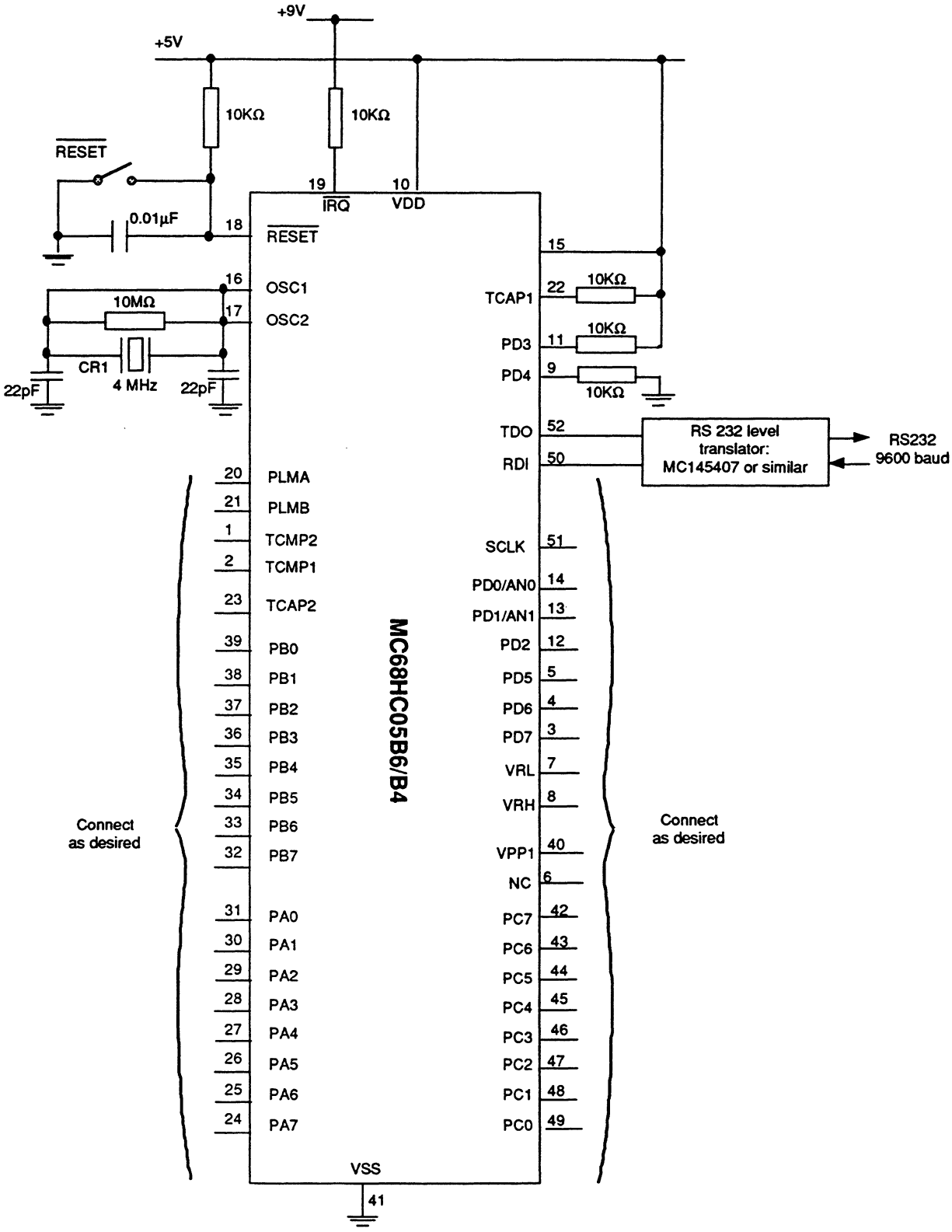

Figure 1. RAM bootstrap schematic

2) BOOTSTRAPPING THE EEPROM1

To bootstrap the EEPROM1 on the MC68HC05B6 in the absence of a built in loader program, use must be made of the RAM bootloader described above. The idea is that an EEPROM1 loader can be written to the users exact requirements then assembled and downloaded into the RAM of the HC05B6 where it will execute and in turn download data and program it into the EEPROM1.

The 6K EEPROM emulation part, the MC68HC805B6, does have a built in EEPROM bootloader in place of the RAM bootloader and there is an accompanying PC program available from Motorola called E2B6 that downloads S-records to the device for programming.

The following is an explanation of an example EEPROM1 bootstrap program for the B6 that has been written to be compatible with the 805B6 PC program E2B6 thus eliminating the need to develop another PC program.

A listing of this program (EE1BOOT) is given at the end of this application note. The MC68HC05B6 has 176 bytes of RAM that can be used for the EEPROM1 bootstrap program, so the protocol must be kept simple and the code written efficiently. The format of the E2B6 program is a transfer of 2 address bytes followed by the data byte that is to be programmed at that location. At the same time the B6 returns the data from the previously programmed location for verification by E2B6. The program EE1BOOT has 4 main sections: a main loop, an erase routine, a program routine and an SCI service routine. The core of both the erase and program subroutines is the extended addressing subroutine EXTSUB which is used to access the EEPROM1 array. This subroutine is built in RAM by the main loop as the address information for the next byte to be programmed is received from the SCI. E2B6 always sends a null character during initialisation which could throw the EE1BOOT program out of synchronisation, as it is already executing before E2B6 is invoked. For this reason EE1BOOT ignores the first character received and treats the second as the first address byte.

The EXTSUB routine is first called as an "LDA $aaaa" to retrieve the last byte programmed for verification. Then the address in the routine is modified as the next address to be programmed is received. When the data byte is received the opcode of EXTSUB is incremented so that it becomes "STA bbbb" before the erase and program routines are called. After programming the opcode is decremented back to LDA before the main loop is repeated.

Note that the EEPROM1 location is always erased before programming. The timer output compare function is used to provide a 10ms delay for erasing and programming and the programming step is skipped to save time if the data presented to that location is $FF. The sequence of events to bootstrap the EEPROM1 of the 68HC05B6 is therefore as follows:

1) Configure the 68HC05B6 as in Figure 1.

2) Assemble the program EE1BOOT and convert it to binary using BINCONV as described in section 1.

3) Set up PC COM port to 9600 baud then release Reset on the HC05B6.

4) Use the command "COPY EE1BOOT.BIN COM1/B" to download EE1BOOT into the RAM of the HC05B6. EE1BOOT will now start to execute.

5) Start the program E2B6 on the PC and follow the instructions to download the desired S-records to the EEPROM1 of the 68HC05B6.

Note:

i) Only the download procedure of E2B6 will work in conjunction with EE1BOOT.

ii) Once the EEPROM1 security bit has been set, the RAM bootloader on the 68HC05B6 will no longer operate. This means that after the device has been reset it will be impossible to download any more data into the EEPROM1 until selfcheck has been executed – selfcheck performs an erase of the entire EEPROM1 array. This means that if the EEPROM1 is to be programmed in several steps, the one that will set the security bit should be done last.

FURTHER POSSIBILITIES

This application note has shown a method for initialising the EEPROM1 on the 68HC05B6 by using the RAM bootloader. It would of course be much simpler to incorporate a EEPROM1 bootloader in the ROM space of the user program, but often there is not enough space. If enough space is available (117 bytes), then EE1BOOT could be incorporated in the application software, thus saving steps 2, 3 &4 in the procedure above.

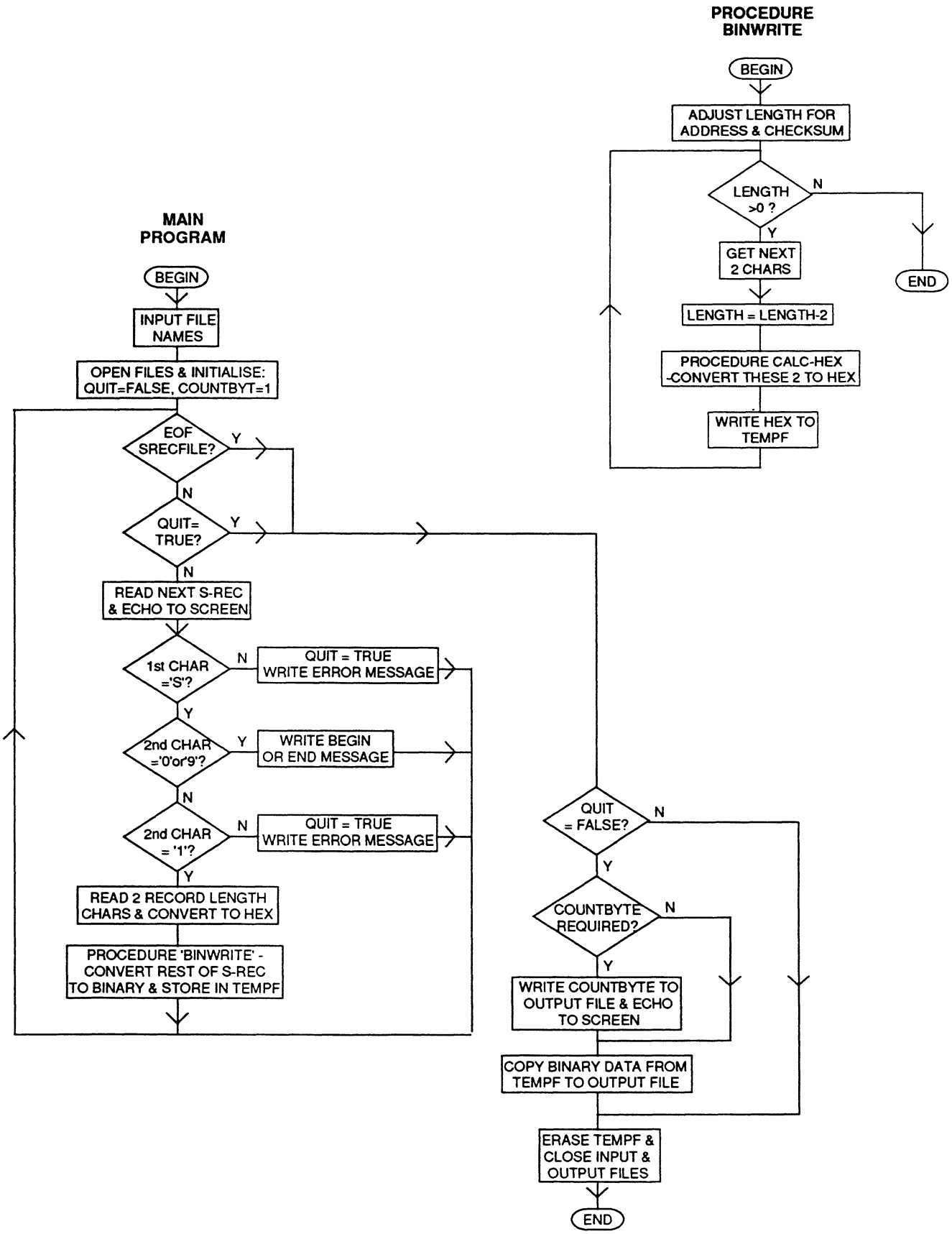

Figure 2. Flow diagram of BINCONV

```
0001                    ********************************************************************************
0002                    *%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*
0003                    *%                                                                              %*
0004                    *%     EE1BOOT - 68HC05B6  EEPROM1 Serial bootloader                            %*
0005                    *%                                                                              %*
0006                    *%     - This prog. is loaded into the RAM of the HC05B6 via the RAM            %*
0007                    *%     bootloader. The program will then start to execute. The format           %*
0008                    *%     has been selected to be the same as that on the 805B6 so that            %*
0009                    *%     the program E2B6 can be used to program the EEPROM1.                     %*
0010                    *% Note: E2B6 sends a null character during initialisaton so this prog          %*
0011                    *%      ignores the first character received on the SCI.                        %*
0012                    *%                                                                              %*
0013                    *%                                                                              %*
0014                    *%     Jeff Wright                Last Updated  10/5/90                         %*
0015                    *%                                                                              %*
0016                    *%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*
0017                    ********************************************************************************
0018
0019
0020                    **********   I/O and INTERNAL registers definition  **********************
0021                    *
0022                    *
0023                    *      I/O registers
0024                    *
0025 0000               PORTA           EQU     $00       port A.
0026 0001               PORTB           EQU     $01       port B.
0027 0002               PORTC           EQU     $02       port C.
0028 0003               PORTD           EQU     $03       port D.
0029 0004               DDRA            EQU     $04       port A DDR.
0030 0005               DDRB            EQU     $05       port B DDR.
0031 0006               DDRC            EQU     $06       port C DDR.
0032
0033 0007               EECONT          EQU     $07
0034 0002               E1ERA           EQU     2
0035 0001               E1LAT           EQU     1
0036 0000               E1PGM           EQU     0
0037
0038 000d               BAUD            EQU     $0D
0039 000e               SCCR1           EQU     $0E
0040 0004               MBIT            EQU     4
0041 000f               SCCR2           EQU     $0F
0042 0010               SCSR            EQU     $10
0043 0005               RDRF            EQU     5
0044 0011               SCDAT           EQU     $11
0045                    *
0046                    *      TIMER registers
0047                    *
0048 0012               TCR             EQU     $12      Timer control register.
0049 0005               TOIE            EQU     5        Timer overflow interrupt enable.
0050 0006               OCIE            EQU     6        Timer output compares interrupt enable.
0051 0007               ICIE            EQU     7        Timer input captures interrupt enable.
0052
```

```
0053 0013                TSR         EQU   $13     Timer status register.
0054 0003                OCF2        EQU   3             Timer output compare 2 flag.
0055 0004                ICF2        EQU   4             Timer input capture 2 flag.
0056 0005                TOF         EQU   5             Timer overflow flag.
0057 0006                OCF1        EQU   6             Timer output compare 1 flag.
0058 0007                ICF1        EQU   7             Timer input capture 1 flag.
0059
0060 0016                TOC1HI      EQU   $16     Timer output compare register 1 (16-bit).
0061 0017                TOC1LO      EQU   $17
0062 0018                TIMHI       EQU   $18     Timer free running counter (16-bit).
0063 0019                TIMLO       EQU   $19
0064
0065                    ***** MISC DEFINITIONS ----
0066
0067 00c6                LDAEXT      EQU   $C6           OP-Code for LDA extended.
0068 0014                MS10        EQU   $14           10mS delay constant.
0069
0070                     *
0071
0072                    ********************************************************************************
0073                     *                                                                              *
0074                     *      START OF CODE                                                           *
0075                     *                                                                              *
0076                    ********************************************************************************
0077
0078 0051                            ORG   $51
0079
0080 0051 a6 00          RESET       LDA   #$00
0081 0053 b7 04                      STA   DDRA          All Ports inputs.
0082 0055 b7 05                      STA   DDRB
0083 0057 b7 06                      STA   DDRC
0084
0085 0059 19 0e          SCIINT      BCLR  MBIT,SCCR1    Initialise SCI - 8 data bits.
0086 005b a6 c0                      LDA   #$C0
0087 005d b7 0d                      STA   BAUD          9600 baud at 4MHz.
0088 005f a6 0c                      LDA   #$0C          Enable transmit and receive.
0089 0061 b7 0f                      STA   SCCR2
0090 0063 b7 10                      STA   SCSR          Clear pending flags.
0091 0065 a6 c6                      LDA   #LDAEXT       Init extended addressing subroutine to LDA.
0092 0067 c7 00 8f                   STA   OPCDE
0093 006a ad 1d                      BSR   SCREAD        Wait here and ignore 1st char (E2B6 init).
0094
0095 006c ad 21          LOOP        BSR   EXTSUB        Load Acc with data from last programmed addr
0096 006e b7 11                      STA   SCDAT         Send it back for host to verify.
0097 0070 ad 17                      BSR   SCREAD        Get high address
0098 0072 c7 00 90                   STA   ADDHI         - and store it.
0099 0075 ad 12                      BSR   SCREAD        Get low address
0100 0077 c7 00 91                   STA   ADDLO         - and store it.
0101 007a ad 0d                      BSR   SCREAD        Get the data to be programmed
0102 007c c7 00 93                   STA   DATA          Store it temporarily.
0103 007f 3c 8f                      INC   OPCDE         Change the ext addr subroutine to STA aaaa.
0104 0081 ad 11                      BSR   ERASEE        Erase the selected address for 10ms.
0105 0083 ad 27                      BSR   PROGEE        Now prog the data for 10mS.
0106 0085 3a 8f                      DEC   OPCDE         Restote ext addr subroutine to LDA aaaa.
0107 0087 20 e3                      BRA   LOOP
0108
```

```
0109                    ********************** SUBROUTINE TO SERVICE SCI ******************************
0110
0111 0089 0b 10 fd      SCREAD      BRCLR   RDRF,SCSR,*
0112 008c b6 11                     LDA     SCDAT
0113 008e 81                        RTS
0114
0115
0116                    ************ EXTENDED ADDRESSING SUBROUTINE TO ACCESS FULL MEMORY MAP ************
0117
0118 008f               EXTSUB      EQU     *
0119 008f 00            OPCDE       FCB     0
0120 0090 00            ADDHI       FCB     0
0121 0091 00            ADDLO       FCB     0
0122 0092 81            RTS
0123
0124 0093 00            DATA        FCB     0                   Reserved Byte for data during erasing.
0125
0126                    ************************** EE1 ERASING SUBROUTINE ********************************
0127
0128 0094 12 07         ERASEE      BSET    E1LAT,EECONT
0129 0096 14 07                     BSET    E1ERA,EECONT
0130 0098 ad f5                     BSR     EXTSUB
0131 009a a6 14                     LDA     #MS10
0132 009c 10 07         DEL1        BSET    E1PGM,EECONT
0133 009e b7 19                     STA     TIMLO               Set up timer for a 10ms count
0134 00a0 b7 16                     STA     TOC1HI
0135 00a2 b7 13                     STA     TSR                 - using output compare 1 function.
0136 00a4 b7 17                     STA     TOC1LO
0137 00a6 0d 13 fd                  BRCLR   OCF1,TSR,*          Wait here for end of erase time
0138 00a9 3f 07                     CLR     EECONT              - erase finsished.
0139 00ab 81                        RTS
0140
0141                  ********************** EE1 PROGRAMMING SUBROUTINE ********************************
0142
0143 00ac 12 07         PROGEE      BSET    E1LAT,EECONT
0144 00ae b6 93                     LDA     DATA
0145 00b0 ad dd                     BSR     EXTSUB
0146 00b2 4c                        INCA
0147 00b3 27 0f                     BEQ     SKIP                Skip programming if data = $FF
0148 00b5 a6 14                     LDA     #MS10
0149 00b7 10 07         DEL         BSET    E1PGM,EECONT
0150 00b9 b7 19                     STA     TIMLO               Set-up timer for 10mS count
0151 00bb b7 16                     STA     TOC1HI
0152 00bd b7 13                     STA     TSR                 - using output compare 1 function.
0153 00bf b7 17                     STA     TOC1LO
0154 00c1 0d 13 fd                  BRCLR   OCF1,TSR,*          Wait here for programming to finish.
0155 00c4 3f 07         SKIP        CLR     EECONT
0156 00c6 81                        RTS
```

```
{*************************************************************************}

program BINCONV; { Program to convert Motorola S-record files to
                binary format. Optional inclusion of a count byte for
                 HC05B6 RAM bootloader etc}
                { Programmer - Jeff Wright, MCU applications
                               Motorola
                               East Kilbride}

{                              Last Updated 10/5/90}

{*************************************************************************}

var
  SrecFile : text;
  BinFile  : file;
  Tempf    : file;
  srec     : string[100];
 Transfer : array[1..20000] of char;
 numread, numwritten : word;
  answer   : char;
  fnamei   : string[15];
  fnameo   : string[15];
  bytout : char;
  countbyt : integer;
  datcnt : integer;
  datval : integer;
  point  : integer;
  cnt1   : integer;
  cnt2   : integer;
  quit   : boolean;
  Count  : boolean;

{-------------------------------------}

 Procedure Calc_hex(chr1,chr2 : integer);

 {Combines 2 characters into a single byte value i.e A5->165, error
  signaled if non hex character detected}

  Begin
  Case chr1 of
  48..57 : chr1 := chr1 - 48;
  65..70 : chr1 := chr1 - 55;   { Is this a valid hex character?}
  else
      begin
         writeln ('invalid data -  conversion aborted');
           quit := true
      end
  end;
  Case chr2 of
  48..57 : chr2 := chr2 - 48;
  65..70 : chr2 := chr2 - 55;
  else
      begin
         writeln ('invalid data - conversion aborted');
           quit := true
      end
```

```
   end;
   datval := chr1*16 + chr2;      {Convert to single byte}
   end;

{-------------------------------------}

   Procedure Binwrite(length,dpoint : integer);

{Converts an S-record line to hex and stores it in a temporary file}

   begin
   length := length-3;        {Allow for address and checksum bytes}
   countbyt := countbyt+length;      {Update running byte total}
   length := length*2;        {Twice as many characters as bytes}
   while length > 0 do
   begin
        cnt1 := Ord(srec[dpoint]);    {Get the next two characters}
        cnt2 := Ord(srec[dpoint+1]);
        dpoint := dpoint+2;            {Update pointer and length}
        length := length-2;

        Calc_hex(cnt1,cnt2);      {Convert two characters into single byte}
        bytout := Chr(datval);    {- now convert that single byte into a }
        blockwrite (tempf,bytout,1)      {character and save it in temporary file}


    end
    end;

{******************  MAIN PROGRAM STARTS BELOW ************************}

begin
     writeln ('S-record to Binary conversion utility');
     writeln;
     writeln;
     write('Input S-record file name? -> ');
     readln(fnamei);
     assign(SrecFile, fnamei);
     write(' Binary output file name? -> ');
     readln(fnameo);
     assign(BinFile, fnameo);
     assign(tempf, 'temp.tmp');
     quit := false;
     countbyt := 1;
     Reset(SrecFile);          {open the two }
     Rewrite(BinFile,1);      {  -selected files}
     Rewrite(tempf,1);      { + a temporary file}
```

```
      while not Eof(SrecFile) and not quit do
      begin
           readln(SrecFile, srec);   {read S-rec into char string srec}
           writeln(srec);

           If srec[1]='S'then        {If string does not start with S then quit}
              begin
                   CASE srec[2] of
                   '1' :             {If not S1 record then loop back}
                       begin
                            cnt1 := Ord(srec[3]);   {get the 2 record length}
                            cnt2 := Ord(srec[4]);    {characters}
                            calc_hex(cnt1,cnt2); {func to produce hex in
                                                  datcnt from cnt1 & 2}
                            datcnt := datval;
                            point := 9;             {point to first data character}
                            binwrite(datcnt,point)  { convert the data in this s-rec
                                                line to binary and store in temp file}
                       end;
                  '0' : writeln ('Conversion started');
                  '9' : writeln ('last S-record done');
                  else
                      begin   {If not S0,S1orS9 record then abort}
                           quit := true;
                           writeln ('Non standard S-record detected - Conversion aborted')
                      end
                end
             end
             else
                 begin      {If 1st char not an S then abort}
                      quit := true;
                      writeln ('Non standard S-record detected - Conversion aborted')
                 end
      end;

      If quit = false then

{If no errors then copy the temporary file to the output file and add in
 a count byte if required}

      begin
           Reset (tempf,1);
           writeln;
           write ('Do you want a count byte added to start of output file? -> ');
           readln (answer);
           If upcase(answer) = 'Y' then
              Begin
                   writeln ('Total size including count byte = ',countbyt);
                   bytout := chr(countbyt);
                   blockwrite (binfile,bytout,1)
              end;
            repeat
                 blockread (tempf,transfer,sizeof(transfer),numread);
                 blockwrite (binfile,transfer,numread,numwritten);
            until (numread=0) or (numwritten <> numread)
      end;

      close(tempf);
      erase(tempf);     {Finished with temporary file so erase it}
      close(SrecFile);
      close(BinFile)    {Close files before quiting}
end.
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

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

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

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

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

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

Freescale Semiconductor, Inc.