Application Note

Z86E04-Based RS-232 Controlled Moving Message Display

AN003801-Z8X1199

ZiLOG WORLDWIDE HEADQUARTERS • 910 E. HAMILTON AVENUE • CAMPBELL, CA 95008
TELEPHONE: 408.558.8500 • FAX: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Windows is a registered trademark of Microsoft Corporation.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Table of Contents

Acknowledgements

Project Lead Engineer

Bob Bongiorno

Application and Support Engineer

Joe Rovito

System and Code Development

Bob Bongiorno

Z86E04-Based RS-232 Controlled Moving Message Display

This application demonstrates the use of a Z86E04 18-pin microcontroller to control a 7-row by 60-column multiplexed moving message LED display. Because of the number of LEDs (420), a microprocessor or microcontroller with external RAM could be decoded and bit-mapped into a RAM buffer and clocked out at a leisurely pace by moving some pointers around. A 63-character message buffered this way requires a minimum of 378 bytes of RAM storage.

It is possible, however, to display a message of reasonable length using only the Z86E04's internal register space, if the message is kept in ASCII and decoded in real time. By keeping the message in ASCII form, a 63-character message can be stored in only 63 register locations.

The real time constraints are severe: fetching and decoding the ASCII characters column by column, then clocking out a 63-bit serial data stream 250 times per second. This application note demonstrates that the Z8 does just that, while producing a flicker-free display using a modest 8-MHz oscillator frequency.

In addition, the application implements a 9600 bps serial input routine for dynamic updating of the ASCII message.

Discussion

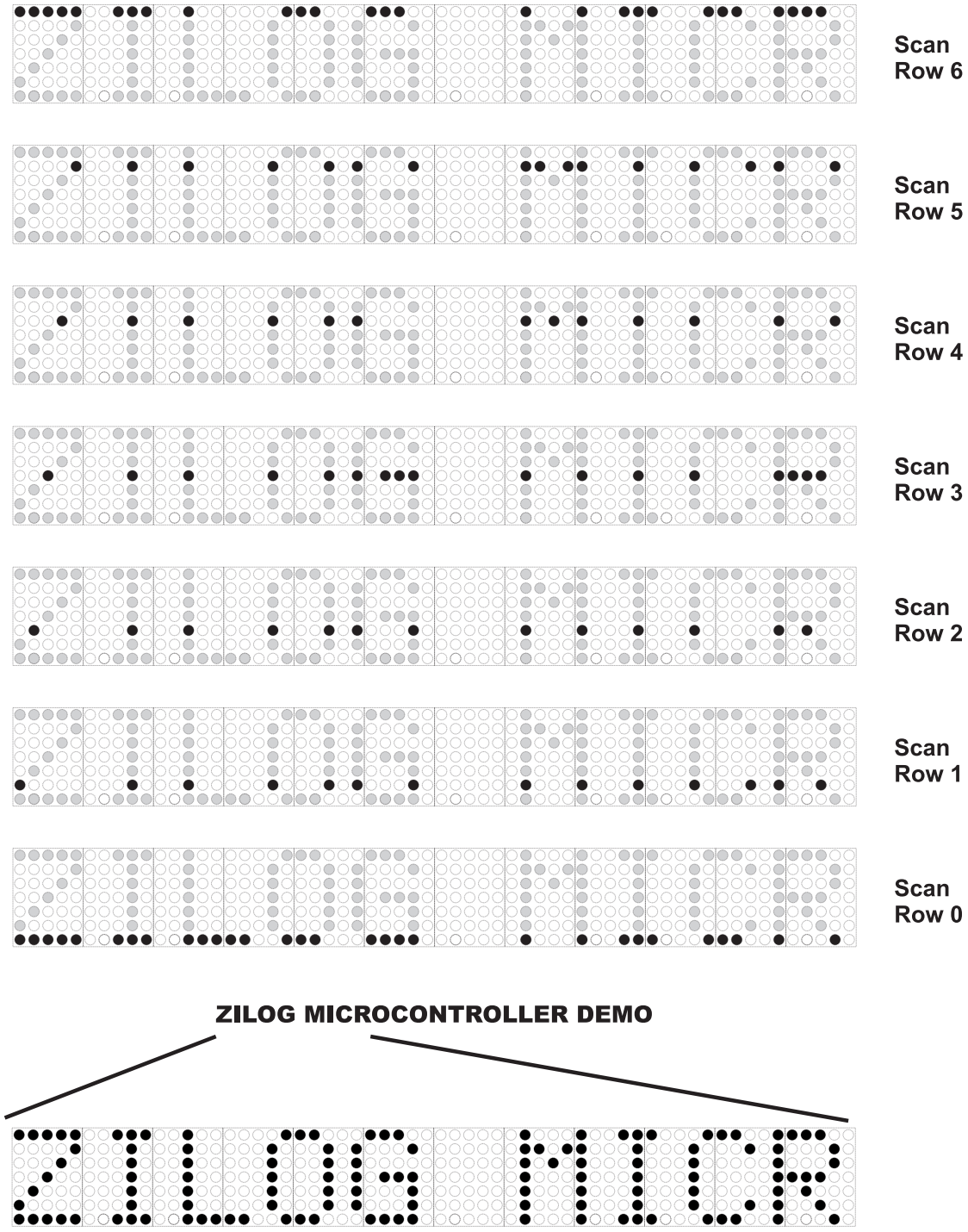
Refreshing a Multiplexed Display

Multiplexing is a common technique used to dramatically increase the number of I/O with only a small increase in hardware. Most designers are familiar with using four inputs and four outputs to scan a 16-key matrix (instead of 16 unique inputs). A multiplexed LED display works the same way, only in reverse.

In this case, the LEDs are electrically arranged to form 7 rows of 60 columns. Each row of the LED is driven for a brief period of time before switching to the next row. Because of the built-in optical integrator in our eyes, known as persistence of vision, switching between rows rapidly enough produces the illusion that all the rows are on at the same time. Figure 1 illustrates a progressive scan example.

To make this work, there are two additional requirements. First, 'overdrive' the LEDs proportionately or they can appear dim. The dimness occurs because any one row is only on 1/7 of the time. And second, update the rows often enough so that each row is scanned at least 40-50 times per second, to reduce the appearance of display flickering. With a red filter in front of the LEDs (required anyway for better contrast), you can update the rows somewhat less often.

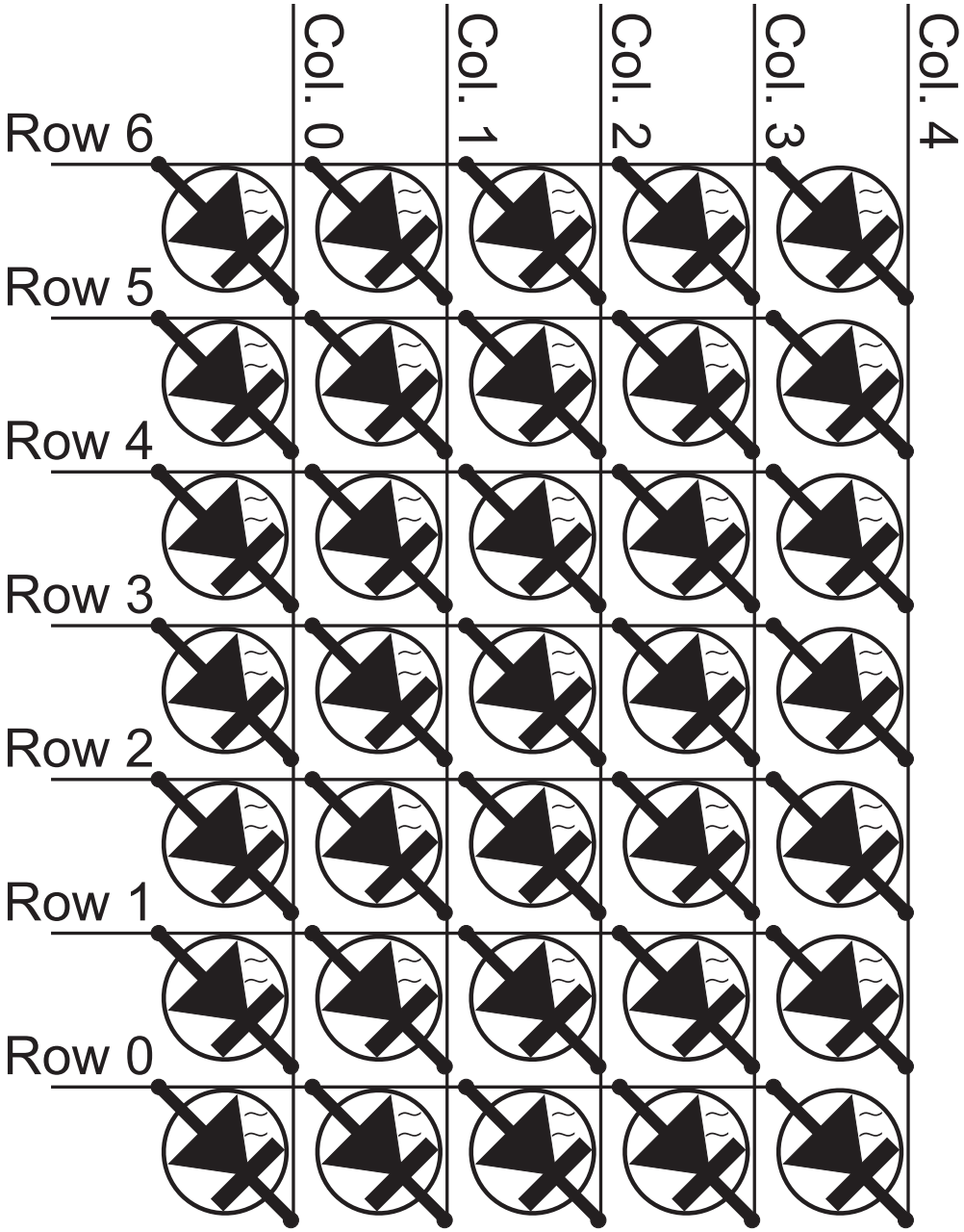
Figure 1. Progressive Scan Example



LED Module Connection

The display modules consist of 5 x 7 LED matrix modules. Each module contains 12 connections: seven rows and five columns, as Figure 2 illustrates.

Figure 2. LED Module Internal Connection



All of the anodes in any single row are connected. All of the cathodes in any single column are also connected. Twelve of these modules are placed left to right to produce a 60-column by 7-row display.

It may not be immediately obvious why a different connection matrix was chosen. For example 21 x 20 requires only 41 outputs instead of the 67. The answer is threefold:

First, because the display is multiplexed, and to get the same perceived brightness, the LEDs must be overdriven by the inverse of the duty cycle, which is typically the number of rows. LEDs can be specified at 15 mA nominal, but 80 to 100 mA at a particular duty cycle. (LEDs may vary, so check the manufacturer's specifications.) This LED specification limits the maximum number of rows to between five and ten. More rows cause a significant decrease in brightness.

Second, low-side drivers are considerably less expensive than their high-side counterparts. For example, the Allegro integrated circuits, each of which drives 32 columns, are only three or four dollars. That price compares to almost one dollar for each of the PNP high-side switches.

Finally, it is easier to keep the number of rows equal to the height of the ASCII character (7), reducing the firmware complexity significantly.

Row and Column Drivers

The seven rows are driven by seven PNP power transistors, wired as saturated high-side switches. Sinking current from the base turns the drivers on hard, thereby powering all of the anodes in a particular row.

The 60 columns are connected to two 32-output low-side drivers, Allegro AN5832. The column information is shifted serially to the drivers, minimizing the wiring and interconnects.

Three of the four unused low-side column drivers perform the row select, driving a 74HC237 (an inverted output `138`). This output turns on a UCN2003 to select the row to drive. This row select process conserves I/O on the Z8 for future expansion—for example, a keypad or an EEPROM.

Displaying a Fixed Message on a Multiplexed Display

Displaying a fixed message is straightforward: decode the message row by row (either ahead of time, if external RAM is available, or in real time as in this application note). Shift out the first row of information and turn the first row on. Delay. Repeat with the second through seventh rows. Then back again to the first.

Displaying and Scrolling a Moving Message

A moving message is more complex. After clocking out and displaying the seventh row, shift the pointers one column to the right and repeat for all seven rows. Then shift another column and display all seven rows. The row scanner is imple-

mented with a timer interrupt. In this application, the timer runs at 250 Hz, producing an individual row refresh of about 36 Hz, but leaving only 4 mS to calculate and clock out the row data.

Serial Input and Command Parser

The application is enhanced with the addition of a serial input routine to update the message display. ZiLOG offers other well-documented application articles detailing bit-banged serial interface routines, so this section is brief.

A 9600 baud receive only interface is implemented. Because of the severe real time constraints in producing a flicker-free display, the display is blanked during reception of serial data. In addition, the first character is ignored, requiring the addition of one dummy character at the start of each message transmitted. This action makes it easier to code the serial routines as well.

Interrupt 2 (the falling edge of P31) detects the start bit of the dummy character, as well as the start bit of subsequent characters. Timer 1, normally used to refresh the display, synchronizes the sampling of the incoming bits.

Operational Result

Command List

The implemented commands are:

- **xLmessage <CR>**
- **xSmessage <CR>**

where:

- **x** is a dummy character signalling the Z8 that serial data is following, acting as a 'start bit' for the sequence. Wait a minimum of 5 mS after sending the dummy character to allow the code to re-synchronize and prepare for the next start bit. All subsequent characters are sent without delays between the characters at the full baud rate of 9600 bits per second.
- **L** or **S** are the commands. **L** is LEFT Shifting Message and **S** is STATIC Message.
- **message** is an ASCII character string, up to 63 characters.
- **<CR>** is the message terminator (ASCII 13 dec).

Display Aesthetics

Some non-obvious aesthetic issues should be considered when displaying moving messages on a multiplexed display.

When displaying a moving message, pad the message with ten ASCII spaces so the message appears to shift in from the right of the display. As the message shifts left, add trailing spaces, or OFFs, to enhance readability. In this application code, the trailing spaces (or “clocked offs”) are added automatically, but the preceding ten spaces should be part of the downloaded ASCII message for left shifting messages.

Scanning *top down* while scrolling left produces the illusion of a character slanted to the left. And vice versa. To experiment, re-arrange the table entries at label ROW_XFORM_TABLE**.**

Produce slightly warped characters by scanning rows 6 0 5 1 4 2 3.

Summary

Controlling a 420-LED moving message display presents severe real time challenges to any microcontroller. This application demonstrates the use of the Z8’s built-in hardware timer, falling-edge interrupt, rich instruction set, and powerful architecture to conquer these challenges.

The application code readily fits into the 1k program space of the Z86E04, using only 640 bytes for code plus another ~ 350 bytes for lookup tables. Expansion to a 2k Z86E08 allows addition of lowercase characters, scores of canned messages, or a more complex command structure without impacting the software beyond the additional feature requirements.

Technical Support

Source Code

Assembling the Application Code

Any Z8 assembler can be used to assemble the code, but ZiLOG Developer Studio (ZDS) is recommended. This integrated suite of software tools allows for program file handling, editing, real-time emulation and debugging when used with the appropriate emulator. Future versions of ZDS incorporates a C-Compiler, simulator, and trace buffer. ZiLOG’s web page contains news and free downloads of the ZDS.

Place all the `.S` files in their own sub-directory. Invoke ZDS and select a new project from the file menu. Under `Target Selection`, select `Family`. Under `Master Select`, select `Z8`. Under `Project Target`, select `Z86E04`. Select the appropriate emulator type to be used. Browse to fill in the project name by clicking on the “...” key. Select the sub-directory containing the `.S` files, name the project, (the extension is added by the program), click `Save` and the first ZDS

screen reappears with the project name, path and file extension filled in. When everything is acceptable, click OK.

Click on the Project tab and select Add to Project, then select Files. Double click on MMSG.S file. This file and all the other.S files are now displayed in the project window. Next, click on the Build tab and select Build. The output window display the assembly results. The standard assembler and linkage settings produce listing and hex files, along with the ZDS files, in the same sub-directory. Save the project and files by clicking on the File tab and selecting these options. The ZDS project file is included, and when ZDS is installed, the above program assembly steps can be omitted.

Programming a One-Time Programmable (OTP) device is accomplished by selecting the OTP option with the hex code installed. Take care to never install the OTP until access to it is required, either for blank checking, verification, or programming. Insert a blank OTP, Z86E04 or Z86E08, into the OTP socket and click on the program OTP selection. Differences exist between earlier GUIs and the ZDS, so take the time to read and understand the operation of the software in use. Pad unused memory locations with FFh before programming. If padding is not consistently done, differences in the check sum occur. The check sum for this program, if unchanged, is 79F6h.

The following source files are included in the source code below:

- mmsg.s The main code
- z8ioass.s Constant definitions and equate
- z8regass.s Z8 internal register assignments
- date.s An embedded date stamp
- ctable.s Character lookup table— ASCII Dots

Instead of displaying each file separately, they are listed exactly in the order and location they are <.included> in the main source, mmsg.s. This order is similar to the way the output listing file (mmsg.lst) is generated.

```
;
;============================================================
;=      TITLE:          mmsg.s                              =
;=      DATE:           started July 1999                   =
;=      PURPOSE:                                            =
;=                                                          =
;=      FILE TYPE:      STAND ALONE MODULE                  =
;=                                                          =
;=      HEADER FILES:   Z8ioass.s,Z8regass.s                =
;=                                                          =
;=      HARDWARE:       NovaTech Z8 Prot. Red Board ZPCB18  =
;=                      Custom Moving Message Display PCB   =
;=                                                          =
;=      ASSEMBLER:      ZiLOG ZDS/ZMASM                     =
;=      PROGRAMMER:     Bob Bongiorno                       =
;============================================================
;
;               release history:
;
;       Version         Date            Description
;       1.00            7/10/1999       Proto release
;       1.10            9/14/1999       Reasonably commented release
;                                       Burn OTP
;
;
;*************************************************************
;IO MAP
;*************************************************************
;P00    ->      STROBE  (Display)
;P01    ->      OE      (Display)
;P02    --      unused
;
;
;P20    ->      DATA    (Display)
;P21    ->      CLOCK   (Display)
;P22    --      unused
;P23    --      unused
;P24    --      unused
;P25    --      unused
;P26    --      unused
;P27    --      unused
;
;P31    --      Rx 5v   TTL RS232
;P32    --      unused
;P33    --      unused
;
            GLOBALS ON        ;Required for Symbol File generation.
;
;*************************************************************
; include files:        non-code generating
;                       constant & i/o assignments - z8ioass.s
;                       register assignments       - z8regass.s
;                       see end of source for character table
;*************************************************************
;
        .INCLUDE "z8ioass.s"
;
;============================================================
```

```
;=      TITLE:          z8ioass.s                           =
;=      DATE:           July 1999                           =
;=      PURPOSE:                                            =
;=                                                          =
;=      FILE TYPE:      .included header file               =
;=                                                          =
;=      HARDWARE:       NovaTech Z8 Prot. Red Board ZPCB18  =
;=                      Custom Moving Message Display PCB   =
;=                                                          =
;=      ASSEMBLER:      ZiLOG ZDS/ZMASM                     =
;=      PROGRAMMER:     Bob Bongiorno                       =
;============================================================
;
;************************************************************
;*                                                          *
;*                     CONSTANT EQUATES                     *
;*                                                          *
;************************************************************
;
ETX             .equ    0
CR              .equ    13
LF              .equ    10
SPACE           .equ    32

b0              .equ    1
b1              .equ    2
b2              .equ    4
b3              .equ    8
b4              .equ    16
b5              .equ    32
b6              .equ    64
b7              .equ    128
;
nb0             .equ    255-b0
nb1             .equ    255-b1
nb2             .equ    255-b2
nb3             .equ    255-b3
nb4             .equ    255-b4
nb5             .equ    255-b5
nb6             .equ    255-b6
nb7             .equ    255-b7
;
STACK           .equ    080h            ;for '08 '04

        .INCLUDE "z8regass.s"
;
;============================================================
;=      TITLE:          Z8REGASS.S                          =
;=      DATE:           July 9, 1999                        =
;=      PURPOSE:        Z8 INTERNAL REGISTER ASSIGNMENTS    =
;=                                                          =
;=                                                          =
;=      FILE TYPE:      .included Header File               =
;=                                                          =
;=                                                          =
;=      HARDWARE:       NovaTech Z8 Prot. Red Board ZPCB18  =
;=                      Custom Moving Message Display PCB   =
;=                                                          =
```

```
;=      ASSEMBLER:      ZiLOG ZDS/ZMASM                 =
;=      PROGRAMMER:     Bob Bongiorno                   =
;=========================================================
;
;*********************************************************
;*                                                       *
;*                  REGISTER EQUATE TABLE                *
;*                                                       *
;*********************************************************
;
;00-03 are hw reserved !
;
P2_IMAGE        .equ    05h
CC_HIGH         .equ    06h
CC_LOW          .equ    07h
ROW_MASK        .equ    08h     ;2^ROW
ROW             .equ    09h
ROW_XFORM       .equ    0Ah
TOD             .equ    0Bh
RXING           .equ    0Ch
RX_CHAR         .equ    0Dh
IN_CHAR         .equ    0Eh
P3_SAMPLE       .equ    0Fh
;
RPWORK          .equ    010h
RP_IRQ          .equ    020h
COMMAND         .equ    030h
MSG_BASE        .equ    031h    ;31 - 6f = 63 char msg buffer
CC_MAX          .equ    070h    ;limit !
;
;*********************************************************
; Interrupt Vectors
;*********************************************************
;
    .ORG 0
;
        .word I0                ; unused
        .word I1                ; unused
        .word I2                ; Falling edge P31 = Start Bit !!!
        .word I3                ; unused
        .word I4                ; unused
        .word I5                ; Timer 1
;
;*********************************************************
; Start of Executed Code
;*********************************************************
;

COLD_RESET:
        jr      START
        nop                     ;space out so time stamp falls on a nice even
        nop                     ;hex boundary
;
;*********************************************************
; Embedded Time/Datestamp and Version
;*********************************************************
;
I_DATE:                                 ;.ASCII "11-24-1991 22:22"
```

```
        .INCLUDE "date.s"
        .ASCII  "09-14-1999 12:24"
;
ROM_VERSION:
        .ASCII  "01.10",CR,LF,ETX       ;00.00 -- 99.99 ASCII
;
;*****************************************************************
; Inline INITZ8  Initialize the monitor S8 processor and  registers
;*****************************************************************
START:
INITZ8:
        DI                      ; initialize interrupt flip flops
        clr     P0              ; required for some older external Rom parts
        clr     IMR             ; make sure nobody's enabled
        clr     IRQ             ; Kill any spurious interrupts
        LD      SPL,#LOW(STACK)
        EI                      ; enable writes to IMR
        DI                      ; disable interrupts
        call    INZ_IO          ; non-inline misc initialization
                                ;
;
        clr     RXING           ;clr RX State Machine ...
RESTART:
        call    INZ_PUP_MSG     ;default moving message
PRESTART:
        call    INIT_TIMER1     ;set up for timer 1 IRQ
        clr     IRQ             ;clr pending
        ld      IMR,#(b5|b2)    ;timer1 & P31 Falling (RS232 RX Start Bit)
;
        ld      CC_HIGH,#0      ;start at 0th character
        ld      CC_LOW,#0       ;0th column
;                               ;RESET  OE      STROBE
;                               ;(b2)   (b1)    (b0)
        ld      P0,#b2          ;1      0       0
        clr     IRQ
      EI
;
;*************************************************************
; Main Foreground execution Loop - wait here while left shifting
;*************************************************************
;
MAIN:
        cp      COMMAND,#'D'    ;stay here until a LEFT shifting msg is done
        jr      NZ,MAIN         ;or indefinitely if a static msg
        DI
        ld      COMMAND,#'L'
        jr      PRESTART
;
;*************************************************************
; This is secondary execution Loop -
; we jump here after Rxing first RS232 char and grab and buffer
; subsequent chars until a <CR> is received
;*************************************************************
;
RS232_FIRST_CHARACTER:
        LD      SPL,#LOW(STACK)
        srp     #RPWORK
        ld      r0,#MSG_BASE
```

```
        ld      r2,#SPACE
$$:
        ld      @r0,r2
        inc     r0
        cp      r0,#(CC_MAX)            ;clr out buffer memory
        jr      NZ,$B
        ld      r2,#CR
        ld      @r0,r2
;
        ld      IMR,#b2
        clr     IRQ
        EI                              ;allow subsequent start bits
;
        ld      r1,#(MSG_BASE-2)        ;first char is -1, pre inc -2
CHARIN_LP:
        inc     r1
        cp      r1,#(CC_MAX+1)
        jr      NZ,$F
        dec     r1                      ;just keep overwriting last char
$$:
        call    CHRIN                   ;get a char in r0, Set Carry iff CR
        ld      @r1,r0
        jr      C,PARSE_NEW_COMMAND
        jr      CHARIN_LP
;
PARSE_NEW_COMMAND:
        DI
        clr     RXING                   ;allow for a NEW msg next time
        jp      PRESTART

CHRIN:
        tm      RXING,#b7               ;WAIT indefinitely for a char
        jr      Z,CHRIN
        ld      r0,RX_CHAR
        and     RXING,#nb7               ;clr char available
        cp      r0,#'Z'+1
        jr      C,$F
        and     r0,#nb5                  ;convert to LC if >= cap 'Z'
$$:
        cp      r0,#CR
        jr      Z,$F
        rcf                              ;no - clear carry
        ret
$$:
        scf                              ;yes set carry
        ret
;
;*************************************************************
;
; End of Foreground execution Loop
;*************************************************************
;
;*************************************************************
;
; SUBROUTINE:   INZ_PUP_MSG
;               Initialize Power Up Message to a default scrolling
;               Message that can be downloaded over.
;               This copies the Message in ROM @ Loc: PUP_MSG
;               to internal register start at Loc: MSG_BASE
;*************************************************************
;
```

```
;
INZ_PUP_MSG:
        ld      COMMAND,#'L'    ;command for a left scrolling msg
        ld      r1,#MSG_BASE    ;point to Start of ASCII MSG in Register Space
;
        ld      r2,#HIGH(PUP_MSG)
        ld      r3,#LOW(PUP_MSG)
;
INZ_PM_LP:
        ldc     r0,@rr2
        cp      r0,#ETX
        jr      Z,$F
        ld      @r1,r0
        inc     r1              ;destination ++
        incw    rr2             ;source ++
        jr      INZ_PM_LP
$$:
        ret
;
PUP_MSG:
        .ASCII  "           HELLO WORLD: ZILOG MICRO MOVING MESSAGE
DEMO",CR,ETX
;
;*************************************************************
; SUBROUTINE:   INIT_TIMER1
;               Set up Timer 1 for Periodic and Recurring
;               I5 IRQs
;*************************************************************
;
INIT_TIMER1:

        and     TMR,#nb3                ;disable count - T1 !
        ld      PRE1,#20*4+(b1|b0)      ;prescal = 20, clock internal, wrap
;
;a pre-scaler of 1 at 8 MHz yields 1uS per T1 !!!
;
        ld      T1,#200         ;4000 uS (200*20) for 8. 0 MHz xtal
                                ;adjust this or prescaler dep. on xtal
                                ;watch service length !
;                               ;this yields a row refresh of 250Hz/7rows
;                               ;~ 35.7 Hz
;               I5 period       Individual Row Refresh  (I5 rate / 7 rows)
;               3000 uS         47.6 Hz
;               3500 uS         40.8 Hz
;               3760 uS         36.0 Hz
;               4000 uS         35.7 Hz
;
;       The more frequently the IRQ, the faster the scan , the less the flicker
;       However, there is a lot to execute in the IRQ (clocking out 60 cols.)
;       and you have to make sure you don't run out of time.
;
;       For flicker free operation, try to keep individual row refresh rate
;       between 40-50 Hz. If you have a red filter, Ok to go down to 30 Hz.
;
;       For much more than 60 columns, or much > 36 Hz, this  requires a
;       faster crystal than the 8 mHZ we are using. Or better code
;       crunching during the DO_60_LOOP:
;
```

```
        or      TMR,#(b2|b3)     ;reload T1 & enable count
        ret
;
;*************************************************************
; Subroutine:   INZ_IO
;               Miscellaneous initialization that does not need
;               to be inline, so it was put in a subroutine to
;               improve code readability.
;*************************************************************
;
INZ_IO:
        LD      IPR,#00001100B  ; Priority:5>3>0>2>1>4 (A>B>C)
        clr     P2M             ; all P2 OUTPUTS
        LD      P3M,#00000001b  ; b7-b2 = 0    RESERVED
                                ; b1   = 1      P31/32 ANALOG !
                                ; b0   = 1      Port 2 push pull
;
        LD      P01M,#00000100b ; b7-b3 = 0    RESERVED
                                ; b2   = 1     RESERVED
                                ; b1b0 = 00    P00-P03 = OUTPUTS
        ld      rp,#RPWORK
        ret
;
;*************************************************************
; Interrupt Routines:
;       I2 is falling edge of P31 (used for RS-232 Rx)
;       I5 is Timer 1 IRQ
;*************************************************************
;
I0:                     ; unused
I1:                     ; unused
I3:                     ; unused
I4:                     ; unused
   JP   START
;
;*************************************************************
; Interrupt Routine:    I2
;       I2 is falling edge of P31 == Start bit
;       we need to delay about 52 uS (1/2 bit time)
;       at 8 mHZ this is 208 cycle to center on Rx data and
;       optionally perform false start bit detection.
;       IRQ latency is about 68 cycles which means we've already delayed
;       quite a bit by the time code at I2: is executed.
;       It also means that there is additional latency when I5 starts
;       interrupting, so the actual target should be ~ 140 cycles
;
;*************************************************************
;
I2:                     ;falling edge P31 = Start bit ...
;                               ;       RESET   OE      STROBE
;                               ;       (b2)    (b1)    (b0)
        ld      P0,#b2          ;10       1      0       0      DISPLAY OFF!
        clr     IN_CHAR         ;?6
        tm      RXING,#b5       ;10
        jr      NZ,I2_SUBSEQUENT;12
 and    P2,#nb6                 ;scope trigger for timing
        and     TMR,#nb3        ;disable any count !
        ld      PRE1,#20*4+(b1  ;prescal = 20, clock internal, 1x only
```

```
;
        ld      T1,#104         ;RESET T1 JIC
        or      TMR,#(b2|b3)    ;reload T1 & enable count !
        clr     IRQ
$$:
        tm      IRQ,#b5
        jr      Z,$B            ;wait 2mS till 1st char flushed thru
;
 or     P2,#b6                  ;10
        ld      RXING,#b5       ;set 1st char HO !
        clr     IRQ
        jp      RS232_FIRST_CHARACTER
;
I2_SUBSEQUENT:
;
;by the time we get around to re-init timer 1 for subsequent bit rate
;IRQs below (see <*****) we are pretty well centered on the incoming bitstream
;having introduced about 176 out of (208-68)  cycles required to be perfectly
;centered. (we are just past center, about 9 uS)
;If desirable, use the scope trigger (firing unused outputs) and
;a delay loop to center perfectly, by moving the reload/restart of T1 back up
;through the code a little. Or carefully count cycles.
;
;This is were you could implement false start bit detect if desired.
;
        and     RXING,#b5        ;10     clr all but b5
        or      RXING,#b6        ;10     set b6
        ld      IMR,#b5          ;10 ONLY timer IRQs allowed
;
        and     TMR,#nb3         ;10  disable any count !
        ld      PRE1,#1*4+(b1|b0);10  prescale = 1, clock internal, wrap
        ld      T1,#104          ;10  RESET T1
        or      TMR,#(b2|b3)     ;10  reload T1 & enable count <*******
 or     P2,#b6                   ;scope trigger
        clr     IRQ              ;clr pending timer/P31 IRQs
;
        iret
;
;*************************************************************
; Interrupt Routine:    I5
;       I5 is the periodic Timer 1 IRQ
;       It is used for two purposes ...
;       I5_SERIAL_IN:
;               If we are in the middle of Receiving RS-232 Data, it times
;               and reads in the accumulates the data bits
;               IRQ latency is about 68 cycles
;       I5:
;               else, it does the Row interlacing/multiplexing and scrolls
;               the message as required.
;*************************************************************
;
;RXING =
;b7  ==> CHAR AVAILABLE in RX_CHAR
;b6  ==> RXING in Progress (IN_CHAR)
;b5  ==> RXING char 2 thru ?
;b4->b0 decimal 0-7 ==> clocking data bits
;b4->b0 decimal 8  ==> stop bits               x1xx 1000 REST STATE MACHINE
;
```

```
;DON'T USE little Rs (working register addressing) - no time to save RP !!
;
I5_SERIAL_IN:
;
  and   P2,#nb6                         ;SCOPE TRIGGER
;
        tm      RXING,#b3               ;still clocking d0-d7 ?
        jr      Z,KEEP_CLOCKING         ;YES
                                        ;NO
        ld      RX_CHAR,IN_CHAR
        or      RXING,#b7               ;set char available
        and     RXING,#nb6              ;leave b5 alone !
        ld      IMR,#b2                 ;re-enable start detection ONLY
        clr     IRQ                     ; reset any pending IRQs
I5_RTNA:
  or    P2,#b6                          ;exit qik - faster than jr DONE_RX
        iret
;
KEEP_CLOCKING:
        scf                             ;assume "1"
        tm      P3_SAMPLE,#b1
        jr      NZ,$F
        rcf
$$:
        rrc     IN_CHAR                 ;rotate into b7 and propagate down
;
NEXT_BIT:
        inc     RXING
I5_RTNB:
  or    P2,#b6                          ;exit qik - faster than jr DONE_RX
        iret
;
ROW_XFORM_TABLE:
        .byte   5,b0                    ;output a "5" to access Row 0
        .byte   3,b1                    ;output a "3" to access Row 1
        .byte   1,b2                    ;output a "1" to access Row 2
        .byte   2,b3                    ;output a "2" to access Row 3
        .byte   6,b4                    ;output a "6" to access Row 4
        .byte   4,b5                    ;output a "4" to access Row 5
        .byte   0,b6                    ;output a "0" to access Row 6
;
;       .byte   5,b0                    ;output a "5" to access Row 0
;       .byte   0,b6                    ;output a "0" to access Row 6
;       .byte   3,b1                    ;output a "3" to access Row 1
;       .byte   4,b5                    ;output a "4" to access Row 5
;       .byte   1,b2                    ;output a "1" to access Row 2
;       .byte   6,b4                    ;output a "6" to access Row 4
;       .byte   2,b3                    ;output a "2" to access Row 3
;;
;Macro definition below ....
;
CLOCK_R0_ROW_MASK:      .macro
                and     P2,#(nb0&nb1)
;               ld      P2,#00000000b   ;clock(b1) = LOW        DATA (b0)=LOW
;               clr     P2
                tm      r0,ROW_MASK     ;if NZ, bit set = LEAVE DATA =0= ON
                jr      Z,$DCRM
;               ld      P2,#00000001b   ;Zero, turn off LED by clocking a one
```

```
                or      P2,#00000001b   ;Zero, turn off LED by clocking a one
$DCRM:
                or      P2,#00000010b   ;Raise Clock
                .endm
;
I5:                                     ;Timer 1 IRQ service routine
;
; Data transferred on CLOCK rising edge
; Latches pass thru to outputs when STROBE is high
; Output is ENABLED when OE is HIGH
;
; LOWER STROBE
; clock out 60 data bits RIGHT TO LEFT:
; Do 60x
;  Lower clock: Output Data: Raise CLOCK;clocks on rising edge
; LOOP
;
; Clock out 3 Row Selects
;
; LOWER OE      ;Display off
; RAISE STROBE, OE
; LOWER STROBE
;
        ld      P3_SAMPLE,P3            ;snapshot P3 NOW ! Just in case RXING
in progress
        tm      RXING,#b6
        jr      NZ,I5_SERIAL_IN         ; yes !!!
;
        push    rp
        srp     #RP_IRQ
        and     ROW,#7                  ;JIC makes a 7
        dec     ROW                     ;For ROW = 6 to 0 Step -1
        jr      PL,$F
        ld      ROW,#6
;
$$:
        ld      r2,#HIGH(ROW_XFORM_TABLE)
        ld      r3,#LOW(ROW_XFORM_TABLE)
        ld      r0,ROW
        rcf
        rlc     r0      ;*2
        add     r3,r0
        adc     r2,#0
        ldc     r0,@rr2
        incw    rr2
        ldc     r1,@rr2
        ld      ROW_XFORM,r0
        ld      ROW_MASK,r1
;
        ld      r4,CC_HIGH
        ld      r5,CC_LOW
        ld      r6,#60                  ;clock out 60 data bits ...
;
        ld      r7,#MSG_BASE
        ld      r8,#CR
        ld      r9,#SPACE
DO_60_LOOP:
;       ld      r0,#MSG_BASE            ;point to Start of ASCII MSG in Registe
```

```
                                        ;Space
        ld      r0,r7
        add     r0,r4                   ;index to proper character
        ld      r1,@r0                  ;fetch
;       cp      r1,#CR
        cp      r1,r8
        clr     r0                      ;assume end of Msg, preload w/ "OFF"
        jr      Z,OUTPUT_DATA           ;CR=YES, skip lookup & increment
;
;       sub     r1,#SPACE               ;CHAR TABLE STARTS AT space
        sub     r1,r9                   ;CHAR TABLE STARTS AT space
        ld      r2,r1                   ;copy to r2
        rlc     r1                      ;*2, carry is clear
       add     r1,r2                   ;*3 (still no overflow if char < 107 dec)
;       rcf
;       clr     r0
        rlc     r1                      ;
        rlc     r0                      ;*6 ...RR0 = (CHAR - 32)*6
;
        add     r1,#LOW(ASCII_TABLE)
        adc     r0,#HIGH(ASCII_TABLE)   ;rr0 points to proper char
        add     r1,r5                   ;CC_LOW
        adc     r0,#0                   ;and now to the proper column
        ldc     r0,@rr0                 ;FETCH Column Data
;                                       ;r0 has column data for Current Column
;
        inc     r5                      ;next Column CC_HIGH,LOW ++
        cp      r5,#6
        jr      NZ,OUTPUT_DATA
        clr     r5
        inc     r4
OUTPUT_DATA:
       CLOCK_R0_ROW_MASK               ;R0 has column data, ROW_MASK has proper
                                       ;bit to ex.
        djnz    r6,DO_60_LOOP
;
DONE_60:
        ld      r0,ROW_XFORM
        ld      ROW_MASK,#b0
        CLOCK_R0_ROW_MASK               ;output LSB of ROW address
;
        ld      ROW_MASK,#b1
        CLOCK_R0_ROW_MASK               ;output MSB of ROW address
;
        ld      ROW_MASK,#b2
        CLOCK_R0_ROW_MASK               ;output HSB of ROW address
;
; after all data is clocked,
; now bang the lines around to strobe the new data in
;
;                                       ;RESET  OE      STROBE
;                                       ;(b2)   (b1)    (b0)
        ld      P0,#b2                  ;1      0       0
        ld      P0,#b2|b0               ;1      0       1
        ld      P0,#b2|b1|b0            ;1      1       1
        ld      P0,#b2|b1               ;1      1       0
;
        cp      ROW,#0
```

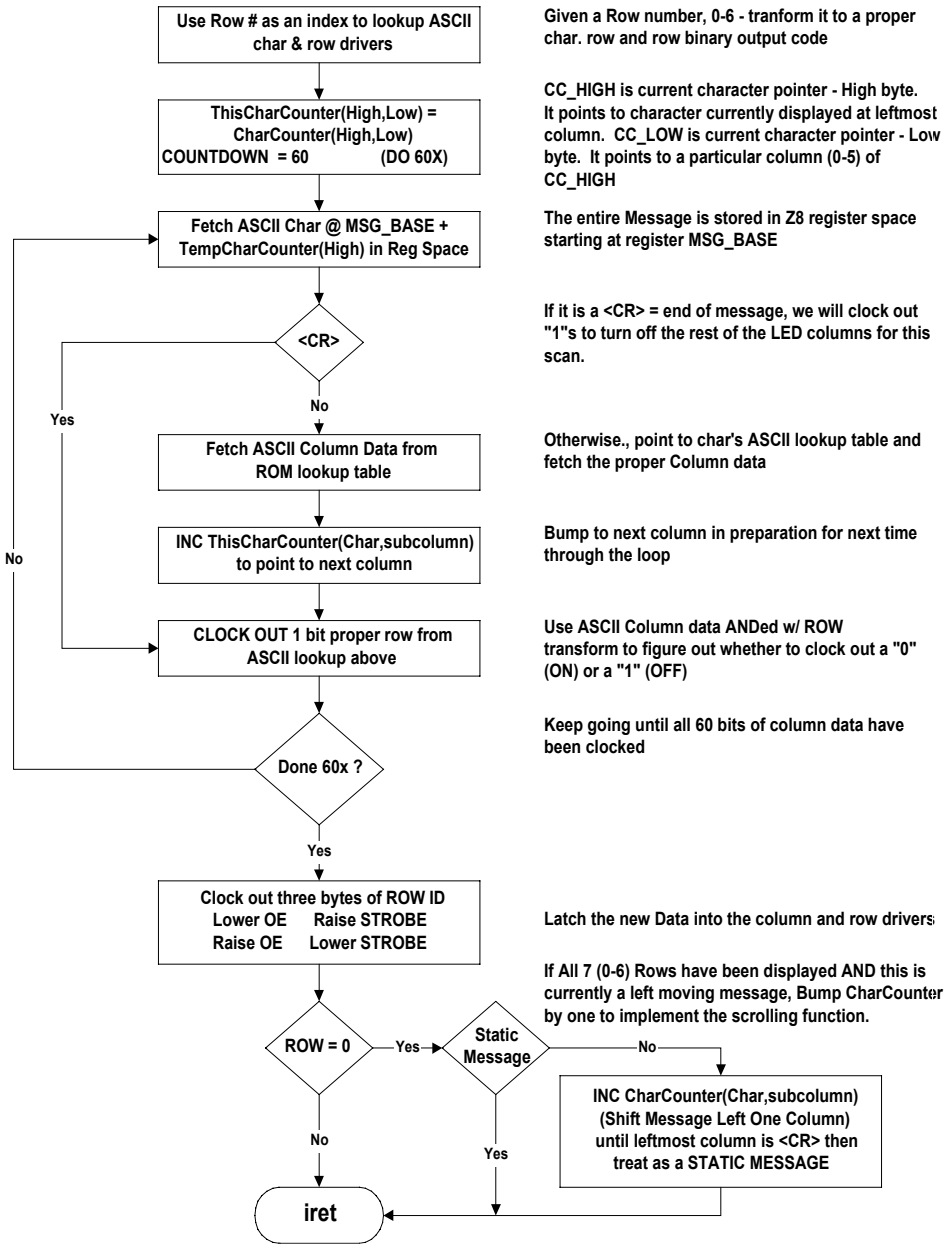
```
        jr      NZ,I5_RTN             ;every time full refresh is complete
        cp      COMMAND,#'S'          ;if Static, leave Pointers alone
        jr      Z,I5_RTN
       cp      COMMAND,#'D'           ;or if shift is done, leave Pointers alone
        jr      Z,I5_RTN
;
        ld      r0,#MSG_BASE          ;point to Start of ASCII MSG in Register
                                      ;Space
        add     r0,CC_HIGH            ;if char @ BASE + CCHI is cr, don't shift
        ld      r1,@r0                ;fetch
        cp      r1,#CR
        jr      NZ,SHIFT_LEFT
        ld      COMMAND,#'D'          ;tell foreground shift left complete !

SHIFT_LEFT:
        inc     CC_LOW                ;else continue scrolling left
        cp      CC_LOW,#6
        jr      NZ,I5_RTN
        clr     CC_LOW
        inc     CC_HIGH
I5_RTN:
   and   IRQ,#nb5
        pop     rp
        iret
;
;*************************************************************
; include file:         character table
;*************************************************************

        .INCLUDE "ctable.s"
   ;
;=========================================================
;=      TITLE:          ctable.s                         =
;=      DATE:           July 1999                        =
;=      PURPOSE:                                         =
;=                                                       =
;=      FILE TYPE:      included char lookup table       =
;=                                                       =
;=      HARDWARE:       NovaTech Z8 Prot. Red Board ZPCB18 =
;=                      Custom Moving Message Display PCB  =
;=                                                       =
;=      ASSEMBLER:      ZiLOG ZDS/ZMASM                  =
;=      PROGRAMMER:     Bob Bongiorno                    =
;=========================================================
ASCII_TABLE:                                            ;Only 32-90 imple-
                                                        ;mented
;
ESPA:   .BYTE   080H,080H,080H,080H,080H,080H           ;20-
EXCLAM: .BYTE   080H,080H,080H,079H,080H,080H           ;21-!
DQUOTE: .BYTE   080H,080H,070H,080H,070H,080H           ;22-"
POUND:  .BYTE   014H,07FH,014H,07FH,014H,080H           ;23-#
DOLLAR: .BYTE   012H,02AH,07FH,02AH,024H,080H           ;24-$
PERCEN: .BYTE   062H,064H,008H,013H,023H,080H           ;25-%
ANDSG:  .BYTE   036H,049H,055H,022H,005H,080H           ;26-&
GRAVE:  .BYTE   080H,080H,050H,060H,080h,080H           ;27-'
LRBRAC: .BYTE   080H,080H,01CH,022H,041H,080H           ;28-(
RRBRAC: .BYTE   080H,041H,022H,01CH,080H,080H           ;29-)
ASTE:   .BYTE   014H,008H,03EH,008H,014H,080H           ;2A-*
```

```
PLUS:   .BYTE   008H,008H,03EH,008H,008H,080H           ;2B-+
COMMA:  .BYTE   080H,080H,005H,006H,080H,080H           ;2C-,
MINUS:  .BYTE   008H,008H,008H,008H,008H,080H           ;2D--
DOT:    .BYTE   080H,080H,003H,003H,080H,080H           ;2E-.
RSLASH: .BYTE   002H,004H,008H,010H,020H,080H           ;2F-/
X0:     .BYTE   03EH,041H,041H,041H,03EH,080H           ;30
X1:     .BYTE   080H,021H,07FH,001H,080H,080H           ;31
X2:     .BYTE   021H,043H,045H,049H,031H,080H           ;32
X3:     .BYTE   042H,041H,051H,069H,046H,080H           ;33
X4:     .BYTE   00CH,014H,024H,07FH,004H,080H           ;34
X5:     .BYTE   072H,051H,051H,051H,04EH,080H           ;35
X6:     .BYTE   01EH,029H,049H,049H,006H,080H           ;36
X7:     .BYTE   040H,047H,048H,050H,060H,080H           ;37
X8:     .BYTE   036H,049H,049H,049H,036H,080H           ;38
X9:     .BYTE   030H,049H,049H,04AH,03CH,080H           ;39
COLON:  .BYTE   080H,080H,036H,036H,080H,080H           ;3A-:
SCOLON: .BYTE   080H,080H,035H,036H,080H,080H           ;3B-;
LARROW: .BYTE   080H,008H,014H,022H,041H,080H           ;3C-<
EQUAL:  .BYTE   014H,014H,014H,014H,014H,080H           ;3D-=
RARROW: .BYTE   080H,041H,022H,014H,008H,080H           ;3E->
QUESTI: .BYTE   020H,040H,045H,048H,030H,080H           ;3F-?
AT:     .BYTE   03EH,041H,05DH,055H,05DH,038H           ;40-@
XA:     .BYTE   03FH,044H,044H,044H,03FH,080H           ;41
XB:     .BYTE   07FH,049H,049H,049H,036H,080H           ;42
XC:     .BYTE   03EH,041H,041H,041H,022H,080H           ;43
XD:     .BYTE   07FH,041H,041H,022H,01CH,080H           ;44
XE:     .BYTE   07FH,049H,049H,049H,041H,080H           ;45
XF:     .BYTE   07FH,048H,048H,048H,040H,080H           ;46
XG:     .BYTE   03EH,041H,049H,049H,02FH,080H           ;47
XH:     .BYTE   07FH,008H,008H,008H,07FH,080H           ;48
XI:     .BYTE   080H,041H,07FH,041H,080H,080H           ;49
XJ:     .BYTE   002H,001H,041H,07EH,040H,080H           ;4A
XK:     .BYTE   07FH,008H,014H,022H,041H,080H           ;4B
XL:     .BYTE   07FH,001H,001H,001H,001H,080H           ;4C
XM:     .BYTE   07FH,020H,018H,020H,07FH,080H           ;4D
XN:     .BYTE   07FH,010H,008H,004H,07FH,080H           ;4E
XO:     .BYTE   03EH,041H,041H,041H,03EH,080H           ;4F
XP:     .BYTE   07FH,048H,048H,048H,030H,080H           ;50
XQ:     .BYTE   03EH,041H,045H,042H,03DH,080H           ;51
XR:     .BYTE   07FH,048H,04CH,04AH,031H,080H           ;52
XS:     .BYTE   031H,049H,049H,049H,046H,080H           ;53
XT:     .BYTE   040H,040H,07FH,040H,040H,080H           ;54
XU:     .BYTE   07EH,001H,001H,001H,07EH,080H           ;55
XV:     .BYTE   07CH,002H,001H,002H,07CH,080H           ;56
XW:     .BYTE   07EH,001H,00EH,001H,07EH,080H           ;57
XX:     .BYTE   063H,014H,008H,014H,063H,080H           ;58
XY:     .BYTE   070H,008H,007H,008H,070H,080H           ;59
XZ:     .BYTE   043H,045H,049H,051H,061H,080H           ;5A
;
; OPTIONAL lowercase SET BELOW - not assembled (see .IF) but can be
; if using a larger ROM (for example, Z86E08)
;
 .IF    0
LBRAC:  .BYTE   080H,080H,07FH,041H,041H,080H,0H            ;5B-[
LSLASH: .BYTE   0H                                          ;5C-\
RBRAC:  .BYTE   080H,041H,041H,07FH,080H,080H,0H            ;5D-]
UPVEE:  .BYTE   010H,020H,07FH,020H,010H,080H,0H            ;5E-^
UNDER:  .BYTE   001H,001H,001H,001H,001H,080H,0H            ;5F-_
```

```
ACCENT: .BYTE   040H,020H,010H,080H,0H                          ;60-`
LA:     .BYTE   002H,015H,015H,015H,00FH,080H,0H                ;61
LB:     .BYTE   07FH,009H,011H,011H,00EH,080H,0H                ;62
LC:     .BYTE   00EH,011H,011H,011H,002H,080H,0H                ;63
XLD:    .BYTE   00EH,011H,011H,009H,07FH,080H,0H                ;64
XLE:    .BYTE   00EH,015H,015H,015H,00CH,080H,0H                ;65
LOWERF: .BYTE   008H,03FH,048H,040H,020H,080H,0H                ;66
LG:     .BYTE   018H,025H,025H,025H,03EH,080H,0H                ;67
LH:     .BYTE   07FH,008H,010H,010H,00FH,080H,0H                ;68
LI:     .BYTE   080H,011H,05FH,001H,080H,080H,0H                ;69
LJ:     .BYTE   002H,001H,011H,05EH,080H,080H,0H                ;6A
LK:     .BYTE   080H,07FH,004H,00AH,011H,080H,0H                ;6B
LL:     .BYTE   080H,041H,07FH,001H,080H,080H,0H                ;6C
LM:     .BYTE   01FH,010H,00CH,010H,00FH,080H,0H                ;6D
LN:     .BYTE   01FH,008H,010H,010H,00FH,080H,0H                ;6E
LO:     .BYTE   00EH,011H,011H,011H,00EH,080H,0H                ;6F
LP:     .BYTE   01FH,014H,014H,014H,008H,080H,0H                ;70
LQ:     .BYTE   008H,014H,014H,00CH,01FH,080H,0H                ;71
LR:     .BYTE   01FH,008H,010H,010H,008H,080H,0H                ;72
LS:     .BYTE   009H,015H,015H,015H,002H,080H,0H                ;73
XLT:    .BYTE   010H,07EH,011H,001H,002H,080H,0H                ;74
LU:     .BYTE   01EH,001H,001H,002H,01FH,080H,0H                ;75
LV:     .BYTE   01CH,002H,001H,002H,01CH,080H,0H                ;76
LW:     .BYTE   01EH,001H,006H,001H,01EH,080H,0H                ;77
LX:     .BYTE   011H,00AH,004H,00AH,011H,080H,0H                ;78
LY:     .BYTE   018H,005H,005H,005H,01EH,080H,0H                ;79
LZ:     .BYTE   011H,013H,015H,019H,011H,080H,0H                ;7A
LCBRAC: .BYTE   080H,008H,036H,041H,080H,080H,0H                ;7B-{
BAR:    .BYTE   080H,0H                                         ;7C-|-BLANK BAR
RCBRAC: .BYTE   080H,041H,036H,008H,080H,080H,0H                ;7D-}
TIL:    .BYTE   00CH,012H,03FH,012H,004H,080H,0H                ;7E-~-CENT SIGN
DEL:    .BYTE   0H                                              ;7F-DO NOT USE
  .endif
```

Flow Charts

Figure 3.



Schematics

Figure 4. Z86E04 Microcontroller Schematic

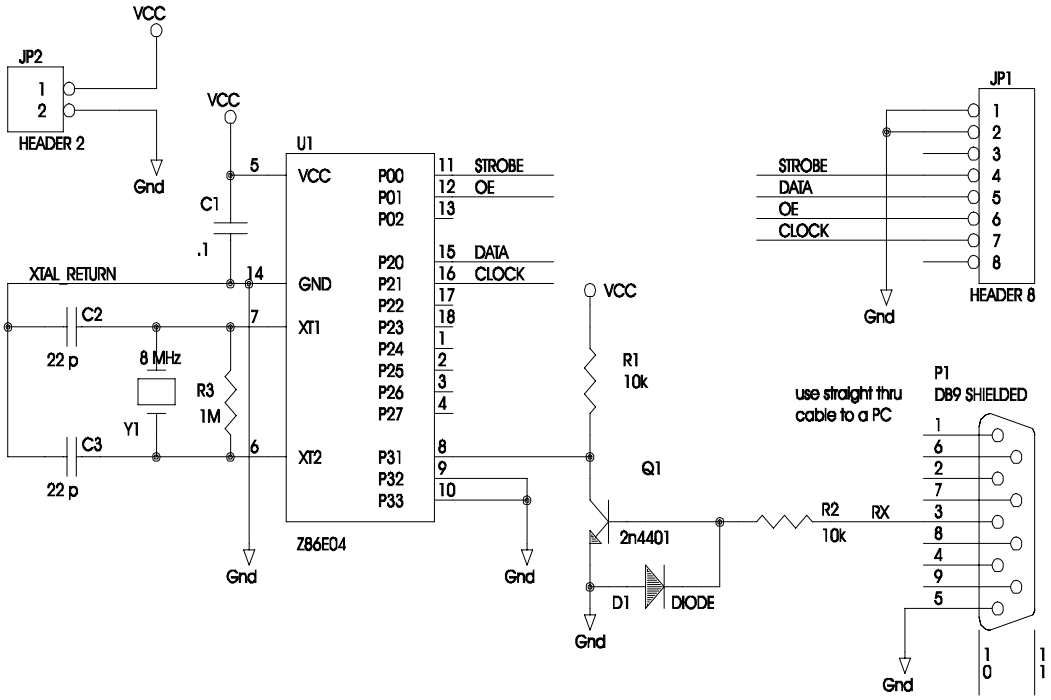
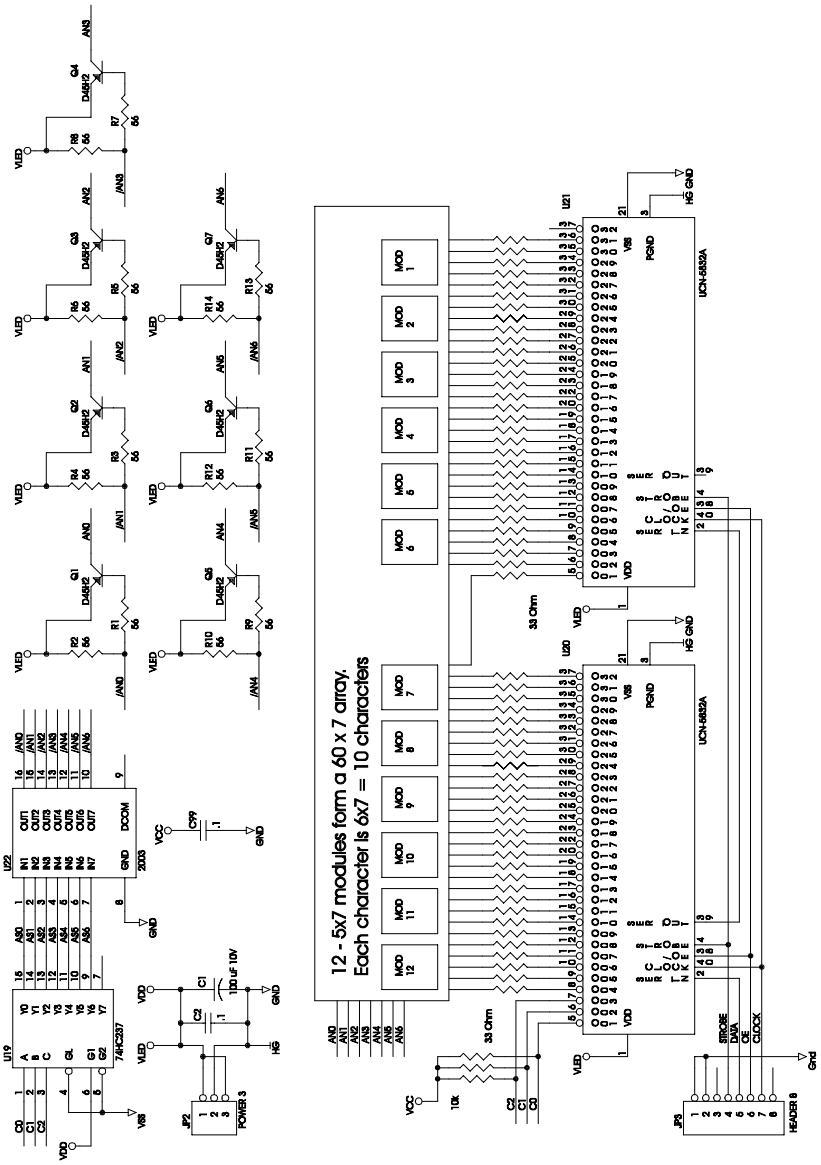


Figure 5. Moving Message Display Driver Schematic



Test Procedure

Equipment Used

Testing the Moving Message Display requires the following equipment:

- Target Moving Message Display PCBs
- 5v, 3A bench supply
- Windows 95/98/NT-based PC with ZDS 2.11 installed
- Z86CCP01ZEM(CCP Emulator) Z86CCP00ZAC (Emulator Accessory Pack)
- 8V @ 0.8 A power supply

The following additional software is required to exercise the display's RS-232 interface:

- `mmsg.bas` BASIC source code for updating Display from `msg.txt`.
- `mmsg.exe` Compile DOS executable of above, hard-coded for COM1.
- `msg.txt` ASCII file used by `mmsg.exe`. Downloaded to test the display
- A DOS or Windows terminal program running on the Com: port of choice, Com. Params = 9600,n,8,1 can be substituted for the above files.

General Test Setup and Execution

Exercise the product by either burning an OTP (stand alone) or running the application from the emulator. (If programming an OTP, and the unused memory is filled with FFh, the check sum should be 79F6h.)

If using an emulator, the PC requires at least two free serial ports: one for the emulator and one for the application's RS-232 interface. Follow the instructions in the Assembling the Application Code section.

Reserve COM1: (if available) for the application serial port. The included BASIC programs are hard-coded for COM1:. If a problem arises, modify the BASIC source, or, instead, type in simple commands using any terminal emulator program on the Com: port chosen.

If using the included BASIC program, run the program and hit <any key> to step through various static and scrolling messages. If running from a terminal program, carefully following the instruction syntax in the Command List section of this application note to type the desired message.

Test Results

Using the included BASIC program or Terminal program, the display of static and scrolling messages can be demonstrated. Modify the source (as described in the

text) to scan `bottom up` instead of `top down`. Also, slow the refresh rate (Timer 1) and observe the loss of smooth animation and onset of flicker.

References

Serial Communications Using the Z8 CCP Software UART, AP96Z8X1300, ZiLOG, Inc., 1997.

Allegro Microsystems Databook, UCN5832A Datasheet, Allegro Microsystems, 1995.