MODIFY YOUR LS-100 RAM-DISK TO ONE MEGABYTE

Mark E. Noneman

In this article, I will discuss a very inexpensive 256K RAM-based Disk Emulator (or RAM-disk) S-100 board. I will then explain a relatively simple modification to this board that will quadruple the available 'disk' memory to 1 Megabyte! This article details methods and techniques used in many electronic designs, so the procedures learned here can also be applied to future electronic projects that you undertake.

LIGHT-SPEED-100 RAM-DISK BOARD

The Light-Speed-100 (LS-100) kit from Digital Research Computers of Duncanville, Texas comes complete with all the discrete components, ICs, sockets, and the Printed Circuit Board (PCB) necessary to complete the project.

The original LS-100 kit had a memory capacity of 256K, and in this article I will show how you can modify it to a full Megabyte. The blank S-100 circuit board for this RAM-disk is still available from Digital Research Computers for $24.95, so even if you do not now own the original board you may still want to build it. For those less adventurous, Digital Research now offers the LS-100 II which is the 1-Megabyte version of the RAM-disk. The price is still a very reasonable $259 for the complete kit, or $309 for the assembled and tested board (including CP/M software drivers). Obviously, if you buy the 1-Megabyte version of the board, you do not need to perform the modification described here, but you may still find some areas of the article to be of interest.

One of the most important features of any kit is the quality of the PCB. This one looks excellent. All the parts are well laid out, especially considering that the board has only two signal layers. The traces of different functions are well separated, and the memory layout appears to follow the recommendations of dynamic RAM manufacturers. This is important since the manufacturers have gone to great pains to determine what two-layer layout and signal routing gives the best noise immunity for their memory products.

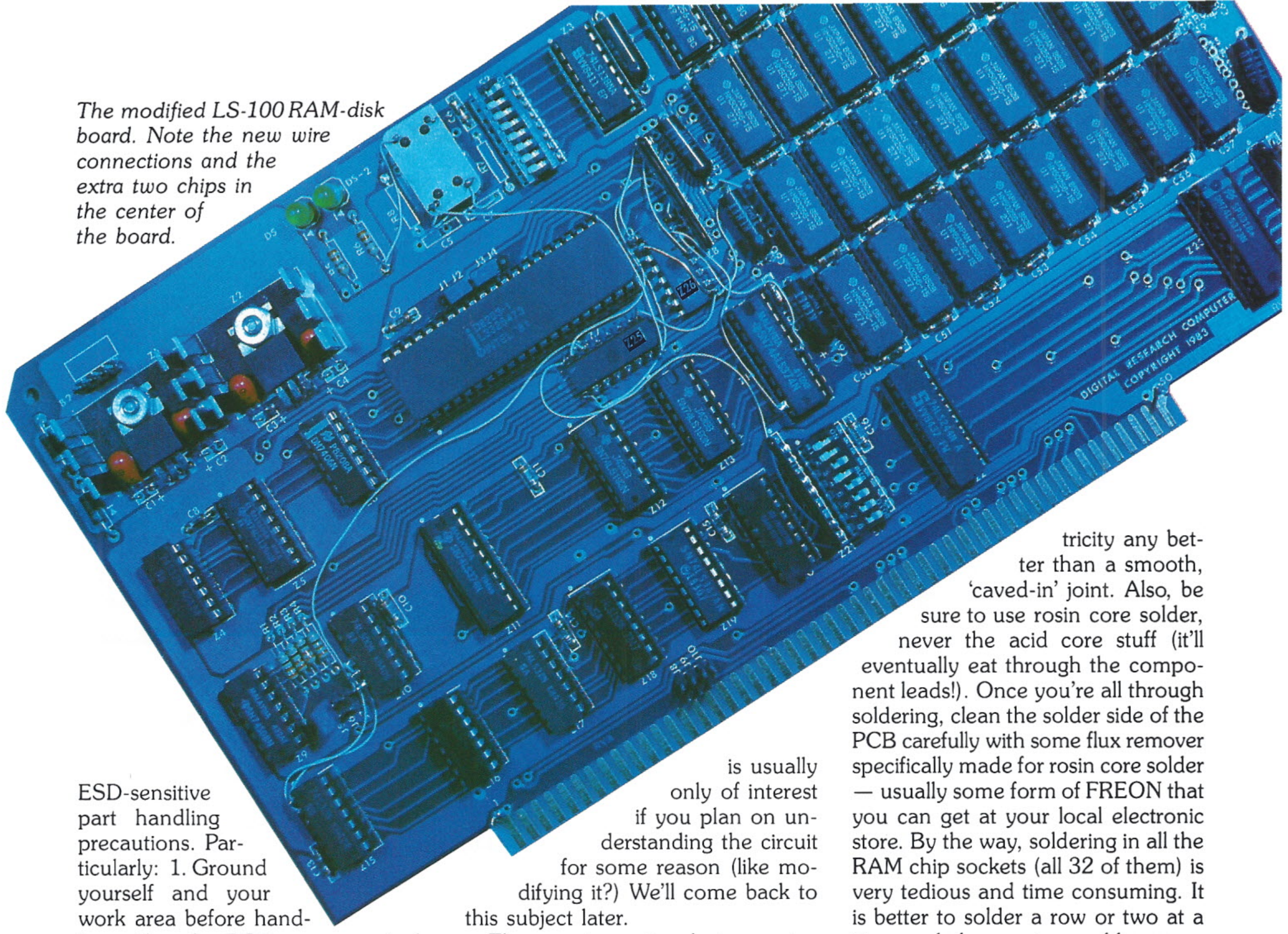
The LS-100 board has a battery-backup option. I haven't tried this feature, but the circuit looks fine from the schematics, and it is definitely a nice feature for those interested. However, you should always save the RAM-disk files to long term media such as a floppy or hard disk.

In summary, the folks at Digital Research Computers have done a great job.

DOCUMENTATION

An important aspect of any kit (or the most important if you have never built a kit before) is the quality of the instructions. If you've done any soldering before and know what a resistor and capacitor look like, the instructions provided with the LS-100 kit should get you through with very little difficulty. However, few practical hints are given. For example, you should be aware of electro-static discharge (ESD). They go to the trouble of shipping all the ESD-sensitive parts in conductive material for a reason: any static charge built up between pins on a device could destroy it. You would think they would warn the kit builder too. If you build this kit, please pay careful attention to ►

Mark Noneman is an Electrical Engineer at TRW, Inc. in the San Diego area. He is a graduate of UCLA School of Electrical Engineering and is experienced in microcomputer and electronic design.


The modified LS-100 RAM-disk board. Note the new wire connections and the extra two chips in the center of the board.

ESD-sensitive part handling precautions. Particularly: 1. Ground yourself and your work area before handling either the PCB or any parts (a cold water pipe is usually a good ground point). 2. Never handle the parts in a carpeted area. 3. Never let someone else touch you or your work area while you're handling ESD-sensitive parts. Remember, just because the parts are installed in the PCB does not mean that they are safe from static discharge. I know many people who have ruined some very expensive parts by poor handling. The memory chips and the 8203 dynamic RAM controller are particularly sensitive.

The manual explains how to setup the board (switches, etc.). This is in general very simple and following the instructions should have you powered up and running in just a few minutes. The only difficult decision to make is between *Normal Ready* and *Advanced Ready*. Just follow the manual and try Advanced Ready. If this doesn't work then switch to Normal Ready. (If this still doesn't work, better start debugging).

The manual also describes the theory of operation for the board. This is usually only of interest if you plan on understanding the circuit for some reason (like modifying it?) We'll come back to this subject later.

There are two minor but annoying problems with the documentation. First of all, the schematic diagram is hand drawn. The boxes and lines are OK, but the handwritten lettering on the signal names and pin numbers can be difficult to read. The second irritation is an inconsistency. The resistor packs for the board are labeled RP1, RP2, and RP3 on the schematic but Z8, Z21, and Z24 on the parts placement diagram and PCB silk-screen legend. For everyone's information, RP1 is really Z21, RP2 is Z8, and RP3 is Z24.

SOME BASIC GUIDELINES

Although the manual gives clear step-by-step insertion instructions, a few suggestions for the uninitiated are in order (those of you who have soldered PCBs before can skip this section). Only use a little solder. Just enough to have a smooth curvature from the solder pad up the component lead. You don't want to gob on the stuff — it won't conduct electricity any better than a smooth, 'caved-in' joint. Also, be sure to use rosin core solder, never the acid core stuff (it'll eventually eat through the component leads!). Once you're all through soldering, clean the solder side of the PCB carefully with some flux remover specifically made for rosin core solder — usually some form of FREON that you can get at your local electronic store. By the way, soldering in all the RAM chip sockets (all 32 of them) is very tedious and time consuming. It is better to solder a row or two at a time and then rest or solder some other sockets on the board. This might help reduce sloppy soldering errors.

When you're through soldering all the discrete parts and sockets, you're ready to install the ICs. Usually, new IC packages have the pins bent outward at about 10 degrees from perpendicular. If you don't have an IC insertion tool and you don't plan to get one, here's what you do before installing the ICs on the sockets: Hold the IC on the edges without the pins (never hold the IC by the pins: static electricity again). Set the row of pins so that they are almost horizontal, points down, on some conductive surface (aluminum foil will work but make sure that both the foil and you are solidly grounded to earth). Now, *very very gently*, rotate the body of the IC so that the entire row of pins gets bent toward perpendicular. Reverse your grip and do the same thing to the other side. The small devices are pretty easily bent. The large ones (like the 8203) take quite a bit of pressure, so be careful. Once the pins are bent under the body of the IC, it's very difficult to straighten them out without breaking them. If you follow this procedure (or use an insertion tool), the chances of bending leads under the body while you're installing the IC into the socket are greatly reduced.

SOFTWARE

There are three major CP/M 2.2 programs that come with the LS-100 board: *DIAG*, *FMAT*, and *INSTALL*. **DIAG** is a software diagnostic program that checks out (rather simply) the operation of the LS-100 board. This is very helpful when you build the board from a kit. This program should indicate whether any parts of the board are not working or, hopefully, that the board seems to be OK.

FMAT is a 'disk' formatting program. Executing this program formats the RAM disk in a way similar to how a floppy disk is formatted (except it only takes a few seconds). The major differences are that a reserved area is formatted with 80H and that no sector header and gap information needs to be written (as is required for floppy or hard disks). The reserved area is used by the INSTALL software to hold a checksum of each sector in the RAM-disk. The data area is filled with E5H just as with a normal floppy or hard disk.

INSTALL is a program which installs the RAM-disk driver just below the CP/M CCP (Console Command Processor). It reassigns certain BIOS and jump assignments so that this installation is almost invisible to calling programs and the RAM-disk can be accessed just like any other disk drive. Unfortunately, you don't get something for nothing. This simple program eats away from the TPA (Transient Program Area) from the top down. This has several unwanted problems, some of which are explained in the manual. Also, problems occur for some compiled programs. For example, when Turbo Pascal compiles a program, it finds the top of the TPA by looking at the BDOS jump address at absolute addresses 6 and 7 and uses this, minus a few hundred bytes, as the base address for the CPU stack. If you use the standard compile (to .COM) options before the RAM-disk is installed and then try to run the

LISTING 1 — PROGRAM Diag1M.PAS
PASCAL PROGRAM TO TEST THE MODIFIED LS-100 BOARD
Program written by Mark E. Noneman — Copyright © 1986 by Mark E. Noneman

```
PROGRAM Diag1M ( INPUT , OUTPUT );

{
   This program tests the LS-100 RAM disk board using pseudorandom
   numbers.  It generates a unique pattern of 2^20 - 1 byte values
   which are written to the disk and then read back to verify that
   the sequence of bytes is the same.  This program takes a very
   long time to run -- over 3 hours on my 4-MHz Z80 machine!
   The assembly language version runs much faster (about 2 minutes).
}

CONST
   MaxMSB = 32;            {count to this minus 1 for MSB register}
   MSBPort = $D1;          {I/O port for MSB register}
   LSBPort = $D2;          {I/O port for LSB register}
   DataPort = $D0;         {I/O port for data}
   BoardMax = 1;           {maximum board number to be entered (7 max) }

TYPE
   ShiftRegister = ARRAY [ 0..19 ] OF BOOLEAN;

VAR
   BoardNum : BYTE;
   Bank : ARRAY [ 0..3 ] OF BYTE;

  PROCEDURE Initialize;

  VAR
    I : INTEGER;

  BEGIN      {Initialize}
    WRITELN;
    WRITELN ( 'Light Speed 100 - One Megabyte Pseudorandom Test Program' );
    WRITELN;
    REPEAT
        WRITE ( 'Enter the board number to test (0-' , BoardMax , ') : ' );
        READLN ( BoardNum )
    UNTIL ( ( BoardNum >= 0 ) AND ( BoardNum <= BoardMax ) );
    BoardNum := BoardNum * 32;
    FOR I := 0 TO 3 DO
         Bank [ I ] := 0
  END;       {Initialize}

  PROCEDURE Clear ( VAR SR : ShiftRegister );    {zero out SR}

  VAR
    I : INTEGER;

  BEGIN      {Clear}
    FOR I := 0 TO 19 DO
        SR [ I ] := FALSE
  END;       {Clear}

  PROCEDURE LFSR ( VAR B : ShiftRegister );
  {  This routine implements a linear-feedback-shift-register
     twenty bits in length of maximal sequency (ie: 2^20-1).
     The formula is: bit 0 = NOT ( bit 16 XOR bit 19 ) while
     bits 1 through 19 are shifted from bits 0 through 18 re-
     spectively.
  }

  VAR
    Temp : BOOLEAN;
    Ptr : INTEGER;

  BEGIN      {LFSR}
    Temp := NOT ( B [ 16 ] XOR B [ 19 ] );
    FOR Ptr := 19 DOWNTO 1 DO
        B [ Ptr ] := B [ Ptr - 1 ];
    B [ 0 ] := Temp
  END;       {LFSR}

  FUNCTION Data ( SR : ShiftRegister ) : BYTE;

  {  Converts the LSByte (BOOLEAN) to binary data. }

  VAR
    B,
    T,
    Ptr : BYTE;
```

program after installing the RAM-disk, you'll get:

Not enough memory
Program Aborted

This happens because INSTALL alters the JMP address. Fortunately, it can be avoided with Turbo Pascal by changing the Options menu to a lower End Address whenever compiling to a .COM file.

By the way, if you hardware reset your computer, the data in the RAM-disk isn't lost. All you do is type INSTALL again to let CP/M know about the RAM-disk, and all your data will still be on that drive — as long as there wasn't any power glitch.

Of course, changing your BIOS will eliminate all of these problems and also negate having to type INSTALL every time you boot up or reset. This method is a little more complex in the short run but much more satisfying once you are done. The instructions in the manual are short but complete, and you shouldn't have any difficulty following their example code. If you have never done any assembly language or BIOS programming before, you'd better get a good book for CP/M beginners.

ONE-MEGABYTE MODIFICATION

The LS-100 is definitely a good product just as it is. But, I wanted to make it better. Sure, 256K of RAM is pretty good, but it's very easy to eat all of that up. How can we make the RAM-disk space bigger? The LS-100 uses 64K-bit dynamic RAM chips. The relatively new 256K-bit dynamic RAM chips come in a compatible package yet increase the number of bits per device by four times. We are going to replace the thirty-two 64K RAMs used in the board with new 256K RAMs. (256K when referring to the chip means 256K *bits*, not bytes.)

Before we go on, however, it's important to understand that the following modification is not approved in any way by Digital Research Computers. In fact, since we are modifying the board to some degree (although we don't have to cut any traces), the warranty is specifically void. I'll try to give you as much information as I can and, if you're careful, this modification should work

```
    FUNCTION Power2 ( X : BYTE ) : INTEGER;

    VAR
      I,
      Temp : INTEGER;

    BEGIN        {Power2}
      Temp := 1;
      FOR I := 1 TO X DO
          Temp := Temp * 2;
      Power2 := Temp
    END;          {Power2}

  BEGIN     {Data}
    T := 0;
    FOR Ptr := 0 TO 7 DO
    BEGIN
      IF SR [Ptr] THEN B := 1 ELSE B:= 0;
      T := T + Power2 ( Ptr ) * B
    END;
    Data := T
  END;       {Data}

  PROCEDURE WriteDisk;

  VAR
    Bits : ShiftRegister;
    MSB,
    LSB,
    Byt : BYTE;

   BEGIN    {WriteDisk}
     Clear ( Bits );    { reset SR -> seed =0 }
     FOR MSB := 0 TO ( MaxMSB - 1 ) DO  {fill for each MSB register}
     BEGIN
       PORT [ MSBPort ] := MSB + BoardNum;
       FOR LSB := 0 TO 255 DO          {fill for each LSB register}
       BEGIN
         PORT [ LSBPort ] := LSB;
         FOR Byt := 0 TO 127 DO      {fill every byte location}
         BEGIN
           PORT [ DataPort ] := Data ( Bits );
           LFSR ( Bits )           {shift data pseudo-randomly}
         END
       END
     END
   END;       {WriteDisk}

  PROCEDURE ReadDisk;

  VAR
    Bits : ShiftRegister;
    MSB,
    LSB,
    Byt,
    D,
    DExpected : BYTE;

     PROCEDURE BadChip ( X , Y : BYTE );

     {   Stores each detected bad bit in the array of banks. }

     BEGIN       {BadChip}
       Bank [ MSB AND 3 ] := ( Bank [ MSB AND 3 ] ) OR ( X XOR Y )
     END;        {BadChip}

   BEGIN       {ReadDisk}
     Clear ( Bits );
     FOR MSB := 0 TO ( MaxMSB - 1 ) DO   {for each MSB register}
     BEGIN
       PORT [ MSBPort ] := MSB + BoardNum;
       FOR LSB := 0 TO 255 DO          {for each LSB register}
       BEGIN
         PORT [ LSBPort ] := LSB;
         FOR Byt := 0 TO 127 DO      {for every byte location}
         BEGIN
           D := PORT [ DataPort ];
           DExpected := Data ( Bits );
           IF D <> DExpected THEN
               BadChip ( D , DExpected );
           LFSR ( Bits )
         END
       END
     END
   END;        {ReadDisk}

  PROCEDURE OutResults;

  VAR
    I : INTEGER;
   BEGIN        {OutResults}
     WRITELN ( 'Test Results:' );
     WRITELN;
     WRITELN ('         D0 D1 D2 D3 D4 D5 D6 D7' );
     FOR I := 0 TO 3 DO
     BEGIN
       WRITE ( 'BANK ' , I , '  ' );
       IF (Bank [I] AND 1) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       IF (Bank [I] AND 2) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       IF (Bank [I] AND 4) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       IF (Bank [I] AND 8) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       IF (Bank [I] AND 16) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       IF (Bank [I] AND 32) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       IF (Bank [I] AND 64) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       IF (Bank [I] AND 128) = 0 THEN WRITE ( 'G  ' ) ELSE WRITE ( 'B  ' );
       WRITELN
     END;
   WRITELN;
   END;         {OutResults}

BEGIN        {program}
   Initialize;
   WRITE ( 'Writing to RAM disk . . . ' );
   WriteDisk;
   WRITELN ( 'Done.' );
   WRITE ( 'Reading from RAM disk . . . ' );
   ReadDisk;
   WRITELN ( 'Done.' );
   OutResults
END.         {program}
```

LISTING 2 — PROGRAM LSCHK.ASM

ASSEMBLY LANGUAGE PROGRAM TO TEST THE MODIFIED LS-100 BOARD

Program written by Mark E. Noneman — Copyright © 1986 by Mark E. Noneman

```
          title     'Pseudorandom RAM-Disk Test'
          maclist   off
;
;    This program tests the LS-100 RAM disk board from Digital
;    Research Computers.  The current version assumes that a minimum
;    of one megabyte of RAM disk memory exists.  The modified board
;    described or four standard boards will qualify.
;
;    The following macros are used to make the program a little more
;    "user friendly."  If you don't have a macro assembler, the only
;    routine that must be coded in is the print macro.  All the others
;    can be left out.
;
version:  .MACRO   ARG1
          jp    varound
          db    ARG1
varound:  equ   $
          .ENDM

print:    .MACRO   ARG1
          push bc
          push de
          push hl
          ld    de,ARG1
          ld    c,9
          call bdos
          pop  hl
          pop  de
          pop  bc
          .ENDM

abort?:   .MACRO
          push af
          push bc
          push de
          push hl
          ld    c,11
          call bdos
          or    a
          jp    nz,exit
          pop  hl
          pop  de
          pop  bc
          pop  af
          .ENDM
          page
;
; This program is designed around the concept of pseudorandom
; testing.  It generates a unique pattern of 2^20 - 1 byte values.
; These are written to the RAM-disk and then reread and compared
```

just fine. But be forewarned, if the board stops working at any time, either from the modification itself or for any other reason, you're on your own. If you bought the LS-100 in kit form, make certain that the board is assembled and in perfect working order before starting this modification.

Back to the drawing board. To understand the following discussion, it would be helpful to have the schematic diagram of the LS-100 board, the specification sheet for the 8203 dynamic RAM controller, a specification sheet for a 64K-bit dynamic RAM chip (any manufacturer will do), and a specification sheet for a 256K-bit dynamic RAM chip (again, any manufacturer). Also, a cursory understanding of the theory of operation of the board, as given in the manual, will be helpful. I will not explain completely how the board works, only what's important to understand for the modification.

DYNAMIC RAM CHIPS

In order to have a good perception of the modification described in this article, we need to examine the requirements of dynamic RAMs. This is not a tutorial on dynamic RAMs, just a look at the features that concern us directly.

Addressing

First of all, most dynamic RAMs have a multiplexed address scheme. That is, the total address space (say 16 bits) is split into two parts (8 bits each — called the ROW address and COLUMN address) that are applied to the chip one at a time. To indicate which of the two parts, control signals are used; usually called Row Address Strobe (RAS*) and Column Address Strobe (CAS*) — the * means an active low signal. This keeps the package pin count smaller but complicates the interface. The 64K RAMs have 16 address bits. The 256K RAMs have 18 address bits. These two extra bits will be multiplexed (like the other bits) onto one package pin. There are several options as to where the two extra address bits will come from; this will be discussed later.

An 8203 dynamic RAM controller

takes care of the 16 address bits for the 64K RAM but has no provisions for the additional two bits required by the 256K RAMs. We'll have to multiplex them ourselves when implementing our modification.

To *address* a dynamic RAM chip, several steps must be taken to either read or write data. First, the row address signals (the eight LSB address bits) are applied to the RAM chip address lines. Then, after an appropriate setup time, the signal RAS* is set low. After a suitable hold time, the other eight address bits are multiplexed to the RAM chip address lines and, after another setup delay, the CAS* line is set low. See Figure 1 (page 70) for a timing diagram of this process. This applies all sixteen address bits to the RAM chip using only 8 address pins. Then, data can be read out of the RAM address, or data can be applied and a write strobe issued. The 8203 handles all of this timing and multiplexing in one package. Notice that the timing diagram for the 8203 in Figure 1 shows at which clock cycles the RAS* and CAS* lines become active. The clock cycle between these two signals is when the address line multiplexing occurs. This scheme provides about 50 nanoseconds (ns) of hold time for RAS* and the same amount for CAS* setup time; more than adequate.

Refreshing

Dynamic RAM chips are so called because they hold the state of each bit (a one or a zero) by the charge on a capacitor and that charge must be *refreshed* periodically before it leaks away. Static RAMs vary in that each bit is held by a transistor configuration called a latch. This latch does not require refreshing but requires at least twice the integrated circuit area and considerably more power than a dynamic cell. That's why dynamic RAM chips are so popular: low power and less circuit board area. The 8203 handles dynamic RAM refreshing.

To refresh the RAM chip, the 8203 controller waits until there is no read or write request on the bus. Then, it applies the next address to be refreshed (held by an internal 8-bit counter) to the RAM address lines and forces

```
; with the original sequence.  This method completely checks out
; the addressing, refreshing, buffering, and RAM chips of the entire
; board.  Any addressing errors would overwrite previously written
; data and show up as an error.
;
; This program continues to run forever, or until any key is pressed.
; To minimize time impact (to check for a pressed key), the macro
; abort? is only run between major events (write, read, output) and
; between MSB register updates (32 times during writes and reads).
;
; The GetBoard routine inputs a valid MSB register number to indicate
; board number.  It really places these bits in the three most signi-
; ficant bits of the MSB value.  The equate BoardMax sets the maximum
; allowed value.  With the standard modification, only two boards can
; exist and BoardMax equals one.  Change this for different mods.
;
;
; Change the following equates for your system and modification.
;

MSBPort:   equ  0d1h       ;MSB address I/O port
LSBPort:   equ  0d2h       ;LSB address I/O port
DataPort:  equ  0d0h       ;data I/O port
MaxMSB:    equ  31         ;maximum value for MSBPort
MaxBank:   equ  4          ;banks 0-3
BoardMax:  equ  1          ;maximum valid MSBAdd value (7 maximum)
           page

lschk:     version    'LSCHK V1.0 13-May-86  (c) 1986 Mark E. Noneman'

           ld   (oldsp),sp      ;save old stack pointer
           ld   sp,stack        ;get new SP


           xor  a               ;clear a and flags
           ld   hl,BytBuf0      ;set pointer
           ld   b,4             ;set counter
clbuf:     ld   (hl),a          ;write data
           inc  hl              ;next byte
           djnz clbuf           ;for b bytes

           print     headmsg

           call GetBoard        ;get MSB value 3 MSBits (0-7)

wrstart:   print     writemsg

           abort?
           call write           ;fill disk

           abort?
           print     readmsg
           call read            ;read entire disk

           call output          ;write out results

           abort?
           jp   wrstart         ;keep going until keypressed

exit:      print     tailmsg

           ld   sp,(oldsp)      ;restore old stack pointer
           ret                  ;normal return

           page
;
;     This routine gets the number between 0 and BoardMax that will
;     be added to the 3 MSB positions of the MSB Register value for
;     the LS-100 board.
;
GetBoard:  print     BdPrompt

           ld   c,1
           call bdos            ;get result
           cp   '0'             ;must be >=0
           jp   m,GetBoard      ;wrong
           cp   BoardMax+'0'+1  ;must be <=0
           jp   p,GetBoard      ;wrong

           and  07h             ;get to binary
           sla  a               ;
           sla  a               ;
           sla  a               ; multiply by 32
           sla  a               ;
           sla  a               ;
           ld   (MSBAdd),a      ;store final number

           ret
;
;     The write routine writes out all 2^20 bytes of data to the RAM
;       disk.  It writes from the maximum MSB value down to zero with
;       256 "sectors" per MSB and 128 bytes per "sector."  At every new
;       MSB value, the keyboard is checked to see if a key has been
;       pressed.  If so, the program aborts to exit.
;
write:      ld   b,MaxMSB
            xor  a          ;reset A and flags
            ld   c,a
            ld   e,a
            ld   h,a
            ld   l,a

WNewM:      ld   a,b                ;get MSByte
            push bc                 ;save b
            push hl                 ;save hl
            ld   hl,MSBAdd          ;get offset value
            add  a,(hl)             ;add to MSByte
            pop  hl                 ;get it back
            out  (MSBPort),a        ;set MSB address byte

WNewL:      ld   a,c                ;get LSByte
            out  (LSBPort),a        ;set LSB address byte
            ld   d,BytPerSect       ;set D counter for byte count

WNewB:      ld   a,e                ;get LSByte
            out  (DataPort),a       ;write data
            call lfsr               ;shift for new pseudorandom number
            dec  d                  ;count off bytes
            jp   nz,WNewB           ;keep going

            inc  c                  ;next LSByte address
            jp   nz,WNewL           ;keep going

            abort?
            pop  bc
            dec  b                  ;next MSByte (down)
            jp   p,WNewM            ;if >=0, go again

            ret
            page
;
;       The read routine reads all of the 2^20 bytes of data from the RAM-
;       disk.  It reads data in the same order as the write routine.  At
;       each read, the data is checked against what it's supposed to be.
;       If it's OK, continue on but if it's wrong, store the bits that are
;       wrong using the routine BadBits.  At every new MSB value, the
;       keyboard is checked to see if a key has been pressed.  If so, the
;       program aborts to exit.
;
read:       ld   b,MaxMSB
            xor  a                  ;reset A and flags
            ld   c,a
            ld   e,a
            ld   h,a
            ld   l,a

RNewM:      ld   a,b                ;get MSByte
            push bc                 ;save b
            ld   (MSBVal),a         ;store MSB in memory for BadBit
            push hl                 ;save hl
            ld   hl,MSBAdd          ;get offset value
            add  a,(hl)             ;add to MSByte
            pop  hl                 ;get it back
            out  (MSBPort),a        ;set MSB address byte

RNewL:      ld   a,c                ;get LSByte
            out  (LSBPort),a        ;set LSB address byte
            ld   d,BytPerSect       ;set D counter for byte count

RNewB:      in   a,(DataPort)       ;read data
            xor  e                  ;=0?
            call nz,BadBit          ;set bad bit if not

            call lfsr               ;shift for new pseudorandom number
            dec  d                  ;count off bytes
            jp   nz,RNewB           ;keep going

            inc  c                  ;next LSByte address
            jp   nz,RNewL           ;keep going

            abort?
            pop  bc
            dec  b                  ;next MSByte (down)
            jp   p,RNewM            ;if >=0, go again

            ret
            page
;
;       The lfsr routine creates a linear-feedback-shift-register twenty
;       bits long of maximal sequence.  This is formed be shifting into the
```

the RAS* line to go low (with the write strobe held false, of course). The CAS* line does not go low, so the RAM never completes its address mode but instead it refreshes the row of capacitors associated with the row address. This process is continued whenever no read or write request is made of the 8203.

Since RAM chips of different sizes have a different number of rows associated with them, each type requires a different number of row addresses to be accessed in order to refresh the entire RAM. Also, a certain time limit — the refresh period — exists for data retention to be guaranteed. For most standard 64K RAMs (the notable exception being Texas Instruments), there are 128 refresh rows (address bits A0 through A6) and a 2-millisecond (ms) refresh period. For all standard 256K RAMs, there are 256 refresh rows (address bits A0 through A7) and a 4ms refresh period. The 8203 on the LS-100 is set up to refresh 256 rows with a 4ms period on address its A0 through A7. The 64K RAMs included (unless TI RAMs are used) ignore bit A7 during refresh. The way the LS-100 is designed, the 8203 will handle all RAM refresh for us without any modification as long as standard (256 rows, 4ms refresh) 256K RAMs are used.

DESIGN OF THE MODIFICATION

As mentioned earlier, the 256K RAMs require two extra address bits which we must supply in order to address the entire memory space. Also, the address line multiplexing must follow the same scheme used by the 8203 so that the RAMs will be properly addressed. In order to do this, we need several items. First of all, we need a clock with the same frequency and phase as that of the 8203. Second, we need a signal which indicates when any of the four RAM-bank RAS* lines is true (each of the four 64K banks has its own RAS* line; these *four* RAS* lines are generated from the *two* 8203 RAS lines using external logic), so that the time to multiplex can be known. Third, we need a multiplexer to mux the two additional address lines.

Fourth, a register is required in order to delay the time of the address line multiplexing, so it lines up with that of the 8203. Finally, some means of applying the multiplexed address signal to all 32 RAM chips must be provided. Figure 2 (page 70) shows a general schematic for these modifications. Figure 3 (page 71) and the following discussion explain the modifications in detail.

Regarding the clock, the 8203 uses an internal crystal oscillator and the LS-100 simply provides an external crystal. It might be possible to apply one of the crystal legs to a driving transistor or a buffer with input hysteresis, but the reliability and stability of the clock would be questionable. A better option is to replace the crystal with a crystal oscillator package of the same frequency (20 MHz). This package provides a very stable, TTL output oscillating signal. In order to do this, the XTAL1 input of the 8203 must be tied to VCC and the XTAL2 input tied to the oscillator source. Even though the crystal oscillator package costs a few dollars (should be less than five dollars anywhere), the extra stability and ease of use is well worth it. We are going to have to route a 20-MHz wire across the PCB and every bit helps.

It turns out that the second item, a RAS*-detector signal, won't cost us anything. There is a spare NAND gate (Z10) on the LS-100 board. Whenever either of the two 8203 RAS lines is low, a RAS* operation is taking place. If we NAND these two lines together, the output will be high whenever any of the four RAM-bank RAS* lines is low.

The third item, the multiplexer, needs to be added to the board. We need a somewhat-fast 2:1 multiplexer. Looking in the data books, it looks like the 74LS157 will work fine. Also, this part should be easily available.

The fourth item, the register, also has to be added to the board. A register capable of 20-MHz operation is required. At first it looks like a 74LS74 might work, but we will see later that a 74S74 is actually required.

Finally, Digital Research Computers kindly provided us with a very simple method of connecting the muxed address line to the RAM chips. On the schematic, the resistor in the resistor network (pin 2 of Z8 or RP2) pulls up

```
;     zero bit the exclusive-NOR of bits 16 and 19 and simultaneously
;     shifting all bits toward the MSB.
;
lfsr:      ld   a,l
           and  1           ;save bit 16
           ld   b,a
           ld   a,l
           and  00001000B ;save bit 19
           rra
           rra
           rra
           xor  b           ;xor them both
           cp   0           ;SEE IF A=0 (COMPARE NOW)
           scf              ;SET CARRY FLAG
           jp   z,shift     ;IF A=0 THEN SKIP CLEAR CARRY
           ccf              ;reset if not equal (X-NOR)
shift:     ld   a,e         ;get LSB
           rla              ;shift
           ld   e,a
           ld   a,h         ;get middle byte
           rla              ;shift
           ld   h,a
           ld   a,l         ;get MSB
           rla              ;shift
           ld   l,a

           ret
           page
;
;     The BadBit routine accepts, in the accumulator, a bit map of bits
;     considered to be bad (xor of read data with good data).  This is
;     OR'ed with the stored map of bad bits (if any) so that a continuous
;     look at bits is kept over the test.  (The bad bit map is NOT reset
;     between runs of the test so that a history of failures can be
;     seen at every print out.
;
BadBit:    push af          ;save a
           exx
           ld   a,(MSBVal) ;get MSB Value
           and  6           ;get bits 2 and 1
           rrca             ;rotate down (divide by 2)

           ld   hl,BytBuf0-1      ;set hl pointer to before buffer
incptr:    inc  hl          ;next position
           dec  a           ;count to bank
           jp   p,incptr    ;keep going if a >= 0

           pop  af          ;get a back
           or   (hl)        ;directly OR with original
           ld   (hl),a      ;save it

           exx              ;get old reg's back

           ret
           page
;
;     The output routine prints out to the screen (console) the bad
;     bit map created during the read routine (via BadBit).  If the
;     bit is a zero, a "G" is printed; otherwise, a "B" is printed.
;
output:    xor  a               ;clear a and flags
           ld   (BankN),a       ;clear bank number
           ld   hl,BytBuf0      ;set pointer

           print     outhead      ;write output header

NextBank:  call WriteBank       ;write out "BANK x  "

           ld   a,(hl)          ;get byte
           ld   b,8             ;number of bits

NewBit:    rra                  ;get LSB into carry
           push af              ;save a
           jp   c,Bad

           print     GoodBit      ;good bit
           jp   ContOut

Bad:       print     BdBit       ;bad bit

ContOut:   pop  af              ;get a back
           djnz NewBit          ;

           print     crlf         ;write cr and lf

           ld   a,(BankN)       ;get bank number
           inc  a               ;next bank
           cp   MaxBank         ;covered all banks?
           ret  z               ;return if done
```

```
        ld   (BankN),a       ;put new bank num back
        inc  hl              ;next bank
        jp   NextBank
        page
;
;    The WriteBank routine simply prints (to the console) out the
;    bank number of the current bad bit map we're going to print in
;    the output routine.
;
WriteBank:    push bc
              push de
              push hl

              print     Bank

              ld   a,(BankN)
              add  a,'0'
              ld   e,a
              ld   c,2
              call bdos

              ld   e,' '
              ld   c,2
              call bdos

              pop  hl
              pop  de
              pop  bc

              ret
        page
;
;       Global equates for program LS100Chk.
;
bdos:   equ  0005h      ;bdos function call address
printf: equ  9          ;print function
cr:     equ  13         ;
lf:     equ  10         ;

BytPerSect:   equ  128  ;number of bytes per disk sector

crlf:   db   cr,lf,'$'

headmsg: db  cr,lf,lf
         db  '                LS-100 Diagnostic Program',cr,lf
         db  '              Test by Pseudorandom Numbers',cr,lf
         db  '$'

BdPrompt: db cr,lf,lf
        db   'Enter the board number to test (0-'
        db   BoardMax+'0',') : '
        db   '$'

writemsg: db cr,lf,lf
        db   ' (Strike any key to abort test.)',cr,lf,lf
        db   'Writing pseudorandom data to disk . . .'
        db   '$'

readmsg: db  ' done.',cr,lf
        db   'Verifying data from disk . . .'
        db   '$'

outhead: db  ' done.',cr,lf,lf
        db   '        D00 D01 D02 D03 D04 D05 D06 D07',cr,lf
        db   '-------------------------------------',cr,lf
        db   '$'

GoodBit: db  ' G ','$'
BdBit:  db   ' B ','$'

Bank:   db   'Bank ','$'

tailmsg: db  cr,lf,lf
        db   ' Program done.',cr,lf
        db   '$'

MSBVal: db   0          ;This is the current MSB register value
MSBAdd: db   0          ;This value is added to MSBVal (0-7 max)
BankN:  db   0          ;This is used to temporarily hold the bank
                        ; number for Output routine.
BytBuf0: db  0          ;This buffer holds the bank bit indicators:
BytBuf1: db  0          ; a one in a bit indicates a bad chip.
BytBuf2: db  0
BytBuf3: db  0

stackbuf: ds 200        ;plenty of stack space
stack:  dw   0
oldsp:  dw   0

        end
```

— END OF LSCHK.ASM LISTING —

pin 1 on all RAM chips. This is because some 64K RAMs have a special method of refreshing that utilizes pin 1; most RAMs don't use this method and their pin 1 is labelled NC. On the LS-100, pin 1 is tied high via the resistor to turn this feature off. Luckily for us, all we need to do is connect our muxed address line to this resistor, at the end that connects to the RAM pin 1 (the A8 address pin on 256K RAMs), and we're all set.

Now, the main question is: Where do the two additional address bits come from? These are the signals marked A18 and A19 in Figure 2. There are many possible solutions, but the one I recommend is to use the two LSB board-select bits. These bits come from Z14 pins 9 and 15 and, in the unmodified LS-100 board, are connected to pins 1 and 2 of Z3. This solution has several advantages. First, it only requires two wires. Second, and perhaps most important, it will simply make this one physical board appear as **four** standard LS-100 logical boards to the software. That's right, virtually no software changes are required with this modification. Only if the driver is installed in your BIOS would a change be required, and that's only the table with the disk parameter information at label DPS(X): (to tell the software that you have 1 Megabyte of RAM-disk).

The only disadvantage with this scheme is that the total possible number of physical boards in a system is reduced to two. And the addressing of the second modified LS-100 board will be a little strange; its address will be four (switch 5) because the software thinks that the first 1-Megabyte physical board is really four logical boards (switches 1 through 4).

The now-unused input pins 1 and 2 of Z3 must be connected to ground for the board to decode addresses correctly. Pins 9 and 15 of Z14 will be connected to the new multiplexer and act as address bits A18 and A19, respectively.

Alternatively, there are ways to use as many as eight boards in a system (such as by connecting pins 5 and 12 of Z14 to pins 1 and 2 of Z3), but all of those options require software changes that will not be covered here.

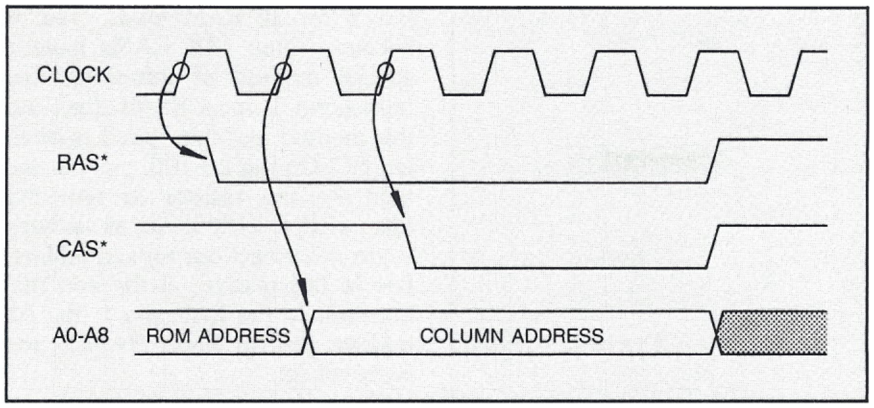


Figure 1. *RAS* and CAS* timing for the 8203.*

TIMING AND LOADING ANALYSIS

Let's analyze the design to see how well it meets timing and loading specifications. An understanding of this section is not required to successfully complete the RAM-disk modification if you follow the method given in this article, but the information is presented here for completeness.

The left side of Table 1 (page 72) shows part of the timing analysis for the modification. Refer to Figure 3 for a complete and final schematic of the modification and as a guide to Table 1. Unfortunately, the 8203 data sheets do not provide complete timing information. Table 1 indicates the 8203 delays necessary to meet timing requirements (and how likely these are). Notice that two problems show up from this analysis: 1) a 74LS74 flip-flop will not get the required setup time, and 2) the setup time for the CAS* signal is hard to count on. In order to resolve these issues, we need to speed up the circuit paths. The easiest way is to replace the 74LS74 flip-flop with a 74S74 (Schottky) flip-flop. This device only requires about 4 ns of setup time and propagation delay of about 10 ns. The right side of Table 1 shows the same timing analysis using the new flip-flop. This change adds an additional 15 ns to the flip-flop setup time and gives us well over 8 ns of CAS* setup time.

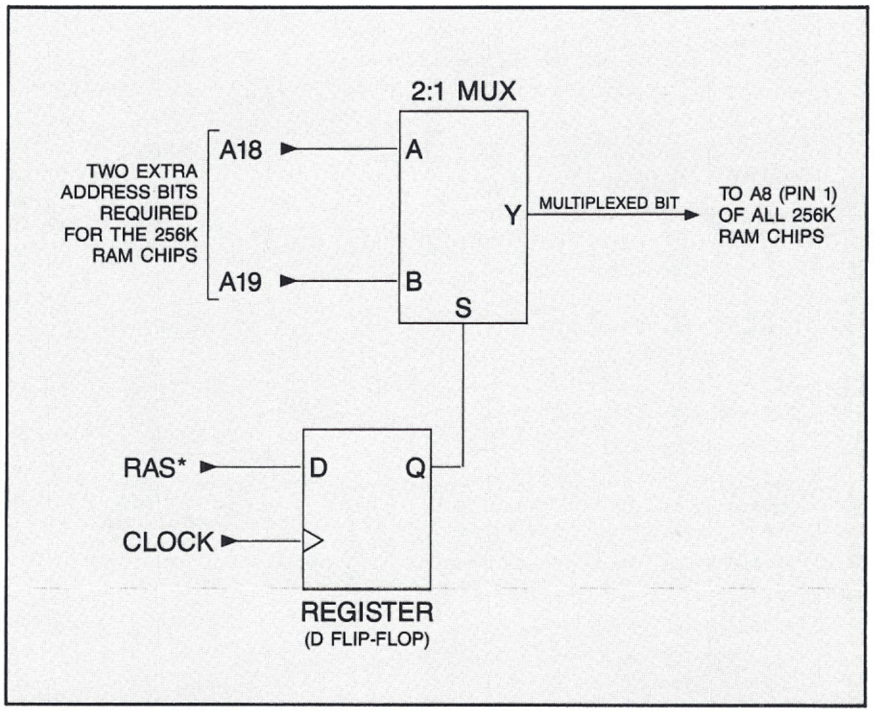
This timing analysis shows how a thorough understanding and some simple mathematics can avoid difficult-to-find errors when troubleshooting. Imagine trying to find out why your RAM-disk keeps losing data because of a setup time violation.

Tables 2 and 3 show a complete loading analysis. The notation used in Table 2 is Zx/y where Zx is the component designator (like Z7, the 8203) and y is the pin number. To determine the Total Load Current, add up the I_{IH}s and I_{IL}s for the parts in the Loading column. These are indicated in Table 3. The Capability column is simply the I_{OH} and I_{OL} of the part on the Pin column. Table 2 shows that there are no loading problems for any of the circuits affected by the modification.

The only other loading analysis concern is the power consumption. We are adding circuits and using RAM chips that consume extra power. The power for the new parts plus the additional power required by the 256K RAM chips draws an extra 200 mA of current from the voltage regulators. The unmodified LS-100 board consumes about 600 mA of current. Thus the total current for the modified board is less than 1 ampere. The two voltage regulators on the LS-100 board can supply about 2 amperes of current nominally. Clearly, there are no power loading issues to be concerned about.

IMPLEMENTING THE DESIGN

Figure 3 shows the complete modification with all of the pertinent pin numbers. First of all, add the oscillator package. This should be done all by itself, so you can be sure that the 8203 is getting the proper clock without adding any other variables. Add the oscillator package by removing XY1 (the crystal), R7, R8, and C5 and C7 if they're installed. Mount the package upside down, over the position of the old crystal, with pin 1 toward the top of the



2:1 MUX
TWO EXTRA ADDRESS BITS REQUIRED FOR THE 256K RAM CHIPS
A18
A19
A
B
S
Y
MULTIPLEXED BIT
TO A8 (PIN 1) OF ALL 256K RAM CHIPS
RAS*
CLOCK
D
Q
REGISTER
(D FLIP-FLOP)

Figure 2. *Simplified diagram of the proposed modification. This circuit generates the additional (multiplexed) address bit required by the 256K chip.*

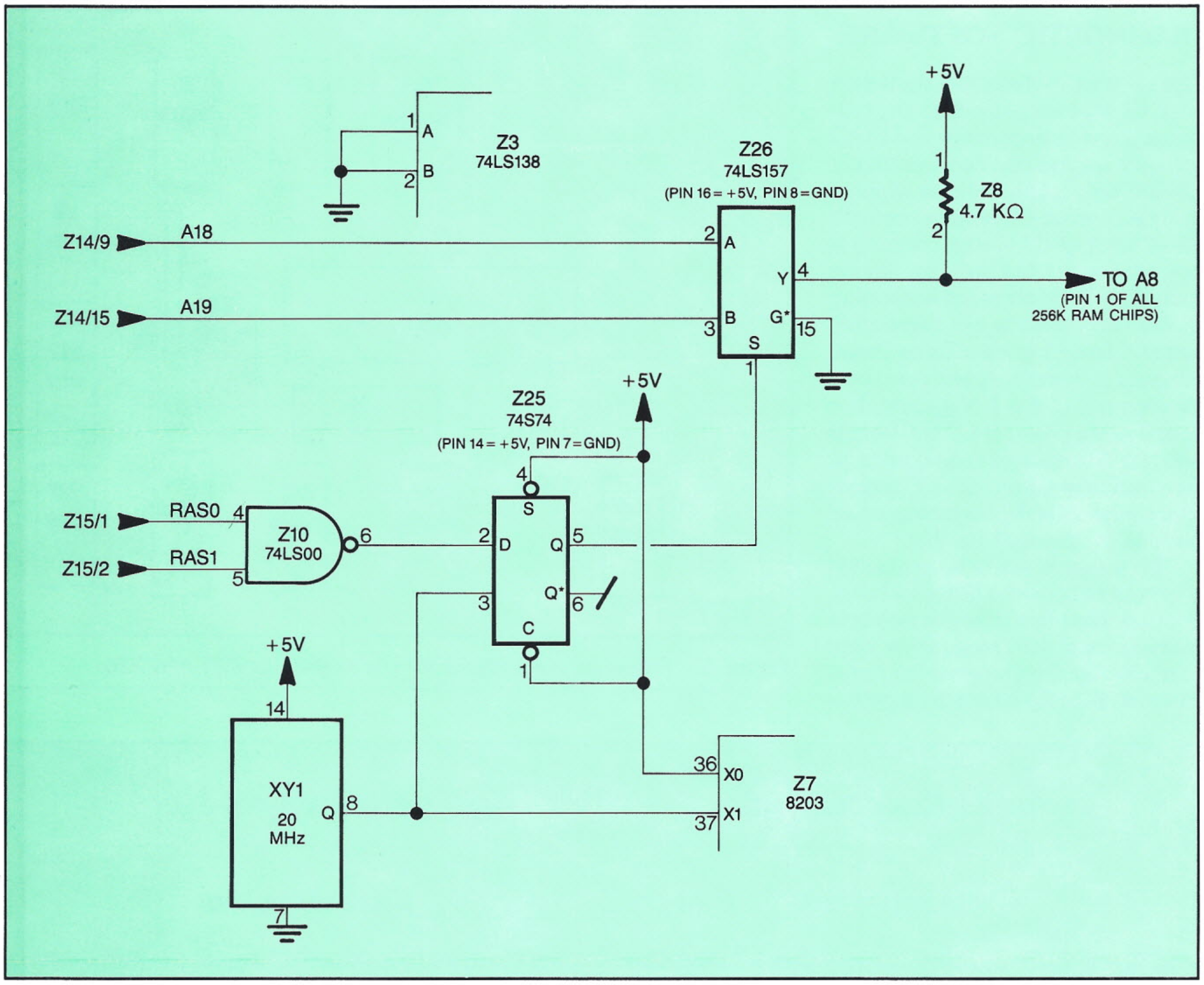


Figure 3. *Detailed schematic of the RAM-disk modification. All pin numbers are indicated here.*

board. Now, wire pin 36 of the 8203 (Z7) to +5V (you can do this on the back of the board). Wire the oscillator to power and ground; pin 14 to +5V and pin 7 to ground. Finally, wire pin 8 of the oscillator to pin 37 of the 8203. These last three connections should be on the component side of the board.

Retest the board, using DIAG, FMAT, and INSTALL, to be sure it operates properly. It should either work fine or not at all. If it doesn't work, you know that something is wrong with the oscillator connections.

Once the oscillator modification works, it's time to add the other parts. The easiest way to add the IC packages is to glue them to the board in a 'dead-bug' position. That is, upside down. Figure 4 shows a suggested parts placement. Use silicon adhesive, epoxy, or even the fast contact glues. The silicon adhesive is nice because it is strong and sets pretty quickly but can be removed if necessary. Cut all of the pins shorter so that they don't stick up like little needles; they are too easily bent and could short together. Cut them to just above the body of the IC (see Figure 5).

I recommend removing Z10, the NAND gate, bending out pins 4, 5, and 6, and cutting them short as well. This avoids cutting traces on the PCB. The same goes for Z14 (the MSB address register), pins 9 and 15. The only connections to be made where neither the pins can be lifted nor traces cut (i.e., the pins must remain in the socket), are to the RAS0 and RAS1 signals. As indicated in Figure 3, solder the wires directly to Z15. Do this by removing Z15 from its socket, carefully solder the wires to pins 1 and 2 up near the body of the IC, and reinsert Z15 into its socket.

Be sure to ground pins 1 and 2 of Z3. You can do this on the back side of the board. Be careful to route all the wires close to the board and with as short a path as is practical. Especially, the wire from the 20-MHz clock and the wire from Z25 to Z26 should be short. Use 30 AWG wire-wrap wire for all the connections.

After completing the connections, replace all the original RAM chips with your new 256K RAMs. Now you're ready to test your modification. Try using DIAG (for boards one through four), FMAT, and INSTALL. Then copy some text files to the RAM-disk. Make them large by concatenating several together into one. Copy to several different file names and try to fill up the disk. Then verify that none of the data is corrupt.

DIAGNOSTIC SOFTWARE

The software routines that started on page 63 will help you debug any problems you may encounter. The DIAG software routine that comes with the LS-100 will roughly indicate whether the board works. However, since each 256K bank of RAM-disk memory is not contiguous but split up among four 256K RAM chips, bit addressing of the two MSBs is not thoroughly tested. Listing 1 gives a Turbo Pascal program to completely test every byte location out of the 1 Megabyte. It is specifically written for LS-100 boards with the 1-Megabyte modification. It won't work for normal LS-100 boards. The constants at the beginning would have to be changed for that.

This program works by generating a unique pattern of 1,048,576 bytes. It writes these to successive disk byte locations and then reads them back to check that the pattern is the same. Since an 8- or 16-bit machine cannot

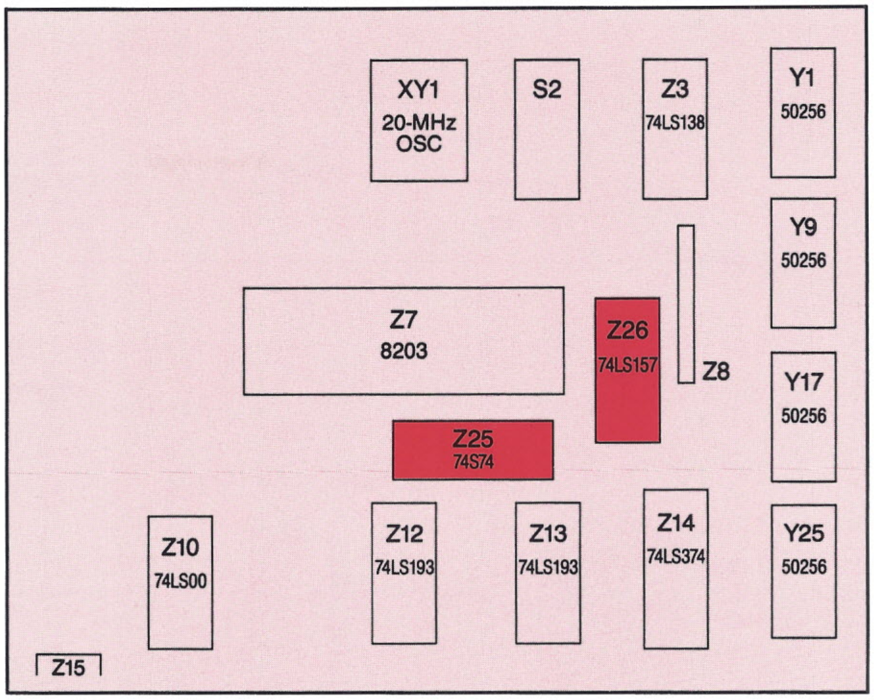


Figure 4. *Proposed parts layout for the new chips Z25 and Z26.*

USING 74LS74 FLIP-FLOP	USING 74S74 FLIP-FLOP
Z25 SETUP TIME: $Z25t_{SU}$ = 50ns − Z7/RAS0,RAS1t_{PD}(max) + Z10t_{PD}(max) $Z25t_{SU}$(required) = 22 ns Z10t_{PD} = 16 ns Therefore, Z7/RAS0,RAS1t_{PD} must be less than 12 ns **This is highly unlikely.**	**Z25 SETUP TIME:** $Z25t_{SU}$ = 50ns − Z7/RAS0,RAS1t_{PD}(max) + Z10t_{PD}(max) $Z25t_{SU}$(required) = 3 ns Z10t_{PD} = 16 ns Therefore, Z7/RAS0,RAS1t_{PD} must be less than 31 ns **This delay should give adequate margin.**
CAS* SETUP TIME: CASt_{SU} = 50 ns − Z25t_{PD}(max) + Z26t_{PD}(max) − Z7/CAS*t_{PD}(min) CASt_{SU}(required) = 0 ns Z25t_{PD} = 28 ns Z26t_{PD} = 30 ns Therefore, Z7/CAS*t_{PD} must be greater than 8 ns **Although likely, somewhat too long to count on.**	**CAS* SETUP TIME:** CASt_{SU} = 50 ns − Z25t_{PD}(max) + Z26t_{PD}(max) −Z7/CAS*t_{PD}(min) CASt_{SU}(required) = 0 ns Z25t_{PD} = 10 ns Z26t_{PD} = 30 ns Therefore, CAS* has a minimum of 10 ns of setup time.

Table 1. *Worst-Case Timing Analysis. A 10% additional derating is included in all times.*

easily produce a 20-bit pseudo-random number in a high-order language, the program simulates one via a Boolean array. The lower eight bits of this array are converted to a binary number and serve as the byte value. Unfortunately, this program is **very** slow (it takes hours) but you should only have to run it once. For those of you who prefer faster testing, Listing 2 gives an assembly language version of the same program. This program uses Z80 mnemonics, so you may have to translate to your own processor's assembly language.

COST ESTIMATE

What is the cost of the described modification? The RAM chips are the most expensive parts by far. At the time of this writing, I.C. Express was selling 150ns (Hitachi) RAM chips for $2.75 each. This is the lowest price I've found. Don't buy RAMs faster than 150ns since the extra speed won't be seen by the 8203. Thirty-two of these RAMs will cost under $100. Try not to get Mostek chips since these appear to have well over 50ns of hold time requirements for RAS*, and they might not work! Beside the RAMs, the two discrete chips (the register and multiplexer) cost less than a dollar each. The crystal oscillator package costs less than $5.

Therefore, for a total of roughly $100, you can quadruple the 256K RAM-disk capacity of the LS-100. Of course, this doesn't include your time, but only you can put a dollar value on that!

LS-100 KIT:	Digital Research Computers PO Box 381450 Duncanville, TX 75138 (214) 225-2309
256K RAMS:	IC Express 15358 Valley Blvd. City of Industry, CA 91746 (800) 882-8181 (CA) (800) 892-8889 (outside CA)
	Microprocessors Unltd, Inc. 24000 S. Peoria Avenue Beggs, OK 74421 (918) 267-4961
ALL PARTS:	JDR Microdevices 110 Knowles Drive Los Gatos, CA 95030 (800) 538-5000

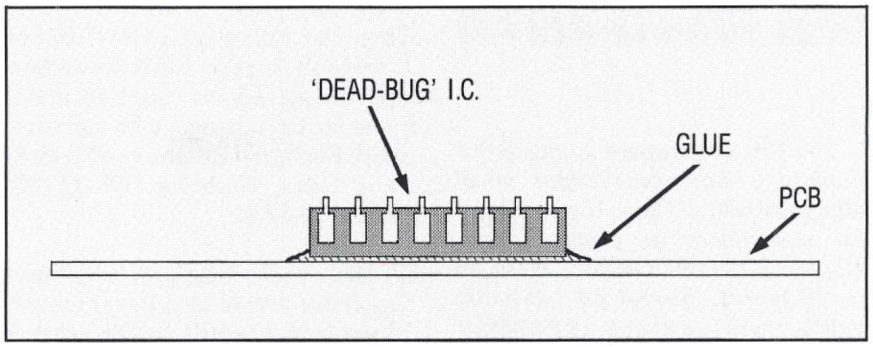


Figure 5. *IC package in a 'dead-bug' position.*

PIN ($I_{OH,L}$)	LOADING ($I_{IH,L}$)	TOTAL (mA)	CAPABILITY (mA)
Z7/21 High	Z15/2 + Z4/13 + Z4/10 + Z10/5	0.17	1
Z7/21 Low		6.4	10
Z7/22 High	Z15/1 + Z4/1 + Z4/4 + Z10/4	0.17	1
Z7/22 Low		6.4	10
Z10/6 High	Z25/2	0.05	0.4
Z10/6 Low		2	8
Z14/9 High	Z26/2	0.02	2.6
Z14/9 Low		0.4	24
Z14/15 High	Z26/3	0.02	2.6
Z14/15 Low		0.4	24
Z25/5 High	Z26/1	0.04	1
Z25/5 Low		0.8	20
Z26/4 High	(-Z8/2) + (Y1 thru Y31)/1	0.32	0.9
Z26/4 Low	(NOTE: Z8 sources current with Z26)	0.32	8
XY1/8 High	Z7/37 + Z25/3	0.14	0.4
XY1/8 Low		6	16

Table 2. *Loading Analysis. Problems only exist if the capability is less than the total load. Thus, the circuits affected by the modification show no problems.*

	Z4,Z15 74S32	Z7 8203	Z8 4.7K	Z10 74LS00	Z14 74LS374	Z25 74S74	Z26 74LS157	Yx 50250	XY1 20MHz
I_{OH}	-1	-1	-.5	-.4	-2.6	-1	-.4	N/A	-.4
I_{OL}	20	10	30	8	24	20	8	N/A	16

I_{IH}	.05	.04	N/A	.02	.02	.05(2)
						.1(3)
							.04(1)
							.02
								.01	N/A

I_{IL}	-2	-2	N/A	-.4	-.4	-2(2) -4(3)	-.8(1) -.4	-.01	N/A

Table 3. *Component Currents (in mA). Pin numbers are shown in parenthesis.*