

DEPARTMENTS

56

Firmware Furnace

66

From the Bench

72

Silicon Update

78

Embedded Techniques

85

ConnecTime

FIRMWARE FURNACE

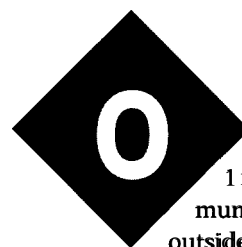
Ed Nisley

Journey to the Protected Land: Fancy Text Output and a Boot Mystery



It's scary
to realize
just how
easy it is

to introduce a system bug so obscure it only shows up under exactly the right combination of circumstances. Ed came across one while showing how to display diagnostic text and it's a beauty.



Our base camp at 1 megabyte communicates with the outside world using 9600-bps serial data and a few LEDs. Before venturing further into the protected-mode wilderness, we need more bandwidth for on-the-scene reports.

A serial link can display relatively slow status information such as a register dump following a system crash. It simply cannot keep up with the torrent of information needed to track the activity of a task-switching operating system. If FFTS has 100 task switches per second, it can send only 10 characters to the serial port during each task!

One of Murphy's corollaries says if you don't display all the information all the time, the most interesting crashes will show the least information.

The Firmware Development Board's Graphics LCD Interface is a better choice for a status display. It doesn't interfere with the PC's BIOS or video-display hardware, and a 640 x 200 panel has lots of room. The only catch is that the CPU must build each character literally dot by dot. This means we should make some timing measurements.

If your embedded application doesn't use the PC video display, of

course, you can show status information on a standard CRT. Should your desk have room for two monitors, you can devote a second video channel to the debugging display. The hardware character generator makes video output much faster than the roll-your-own LCD interface.

This month, we'll add the protected-mode code required for character output to both a VGA board and the Graphics LCD Interface. I'll also explore a mystery in one of the drivers that may save your bacon some day.

ARRANGING THE CHARS

DOS regards even the fanciest video hardware as a glorified Teletype: characters appear one by one starting at the upper left. When the bottom line fills up, a vertical scroll makes room for more text. The speed of that operation contributes mightily to the board's DOS-video benchmark rating. There is no standard way to position the cursor or change text colors without using ANSI control strings.

Fortunately, we use hardware that doesn't have to look like a dimwit terminal. The real-time system-status output will fit neatly on a fixed-format screen which doesn't scroll vertically. Because a program generates all the output text, presumably without typing errors, we don't need even rudimentary character editing. Even better, we can add features as we need them rather than writing a huge lump of code at once.

The status display does require cursor positioning and color control, but the prospect of writing an ANSI command parser in protected-mode assembler gave me pause. Rather than get bogged down in overly complex routines, I opted for a simpler system with binary codes. If you'd like to write a full ANSI decoder, the details are in CA/46!

The code in Listing 1 copies a string to the video display. When the loop detects a V I D C M D byte, it calls the command decoder to interpret the next few bytes in the string. A trailing zero marks the end of the string, an idiom familiar to C aficionados.

Listing 2 is the video-command decoder. The snippet of code following

Listing 1—This routine displays a character string on the video hardware. The loop scrutinizes each byte to locate cursor and color-control sequences as well as the binary zero marking the end of the string. The input parameters are the segment and offset of the string, which allows the string to reside in any valid data segment. A similar routine produces output on the LCD panel.

```

PROC    Vi dSendString
ARG     StrSeg:DWORD,pString:DWORD
USES    EAX,ESI,ES

MOV     EAX,[StrSeg]
MOV     ES,AX
MOV     ESI,[pString]
OR      EAX,ESI           ; skip if pointer is null
JZ      SHORT @@Done

@@Continue:
LOOBS   [BYTE PTR ES:ESI]; fetch the byte
CMP     AL,0             ; check for terminator
JZ      SHORT @@Done

CMP     AL,VIDCMD        ; command byte?
JE      SHORT @@CmdCode  ; yes, special case
CALL    Vi dPutChar,EAX   ; no, show the char
JMP     @@Continue       ; and pick up the next one

@@CmdCode:
CALL    VidCommand       ; yup, invoke command decoder
JMP     @@Continue       ; and pick up next char

@@Done:
RET
ENDP    Vi dSendString

```

each C M P converts the byte after a V I D C M D into a row, column, or attribute. When we need a few more commands, we won't invoke any rocket science to add them, although changing to a table-driven decoder may be a good idea at some point.

This simple command encoding has a gotcha. Setting the cursor to row or column 0 embeds a binary zero in the string. Our command decoder interprets the result correctly because it expects a numeric value rather than a character at that spot. The C-style string routines, which we haven't written yet, will terminate early when they encounter that zero. I opted for a simple solution: the code ignores the high-order bit of the row and column. You can OR each coordinate with 80 hex or just replace each zero with 80.

Remember that only the FFTS kernel will display status and tracing information through this interface. We'll build a more polite routine for user code when we need it.

The process of moving characters and color attributes to the video buffer should be familiar from previous columns as well as your own experi-

ence in real mode. There is one exception—we need a segment descriptor for the video buffer!

GAINING ACCESS, LOSING RISK

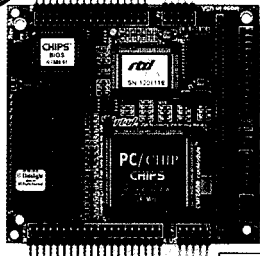
The original PC had two video adapters: a monochrome card (MDA) for decent text and a color card (CGA) for mediocre graphics with ugly text. You could install both cards in the same machine because their I/O ports and memory addresses were different. BIOS and DOS text output went to the "primary" display selected by a system-board switch.

The VGA BIOS ignores that switch. It reads the monitor ID bits through the video cable and sets itself up accordingly. Fortunately, the PC-compatibility barnacles dictate that the BIOS data area must identify the VGA as either an MDAish or CGAish card. The advanced capabilities of the VGA aren't obvious at this level.

The segment descriptor required in protected mode must include the video buffer's starting address and length. The length is easy enough because the BIOS puts the video page size at 0040:004C. A standard 80-

Replace Four Conventional PC/104 Modules with One SuperXT™ CMF8680 cpuModule™

Embedded PC/XT Controller with
Intelligent Power Management



PC/104 Compliant
Actual Size: 3.6 x 3.8 x 0.6"

\$449
100 pcs.

- PC/XT compatibility with 286 emulation
- 14 MHz, 16-bit 8086 CPU
- +5V only; 1.6W at 14.3 MHz, 1 W at 7.2 MHz
- Intelligent sleep modes, 0.1W in Suspend
- ROM-DOS and RTD enhanced BIOS
- Compatible with MS-DOS & real-time operating systems
- 1 M bootable Solid State Disk & free software
- 4K-bit configuration EEPROM (2K for use)
- 2M on-board DRAM
- IDE & floppy interfaces
- CGA CRT/LCD controller
- Two RS-232 ports, one RS-485 port
- Parallel, XT keyboard & speaker ports
- Optional X-Y keypad scanning/PCMCIA interface
- Watchdog timer & real-time clock

Expand This Or Any PC/104 System
with the

CM106 Super VGA Controller utilityModule™

- Mono/color STN & TFT flat panel support
- Simultaneous CRT & LCD operation
- Resolution to 1024 x 768 pixels
- Displays up to 256 colors

\$223
100 pcs.

Speed Product Development with the DS8680 Development System

Your DS8680 includes the CMF8680, CM106 keypad scanning/PCMCIA, CM104 with 1.8" 85MB hard drive, CM106 SVGA controller & DM5406 12-bit, 100 kHz dataModule™ in an enclosure with external power supply, 3.5" floppy, keyboard, keypad, TB50 terminal board, SIGNAL*VIEW™, SIGNAL*MATH™, MS-DOS, SSD software & rtdLinX™ for just \$2950.

For more information on our PC/104 and ISA bus products, call today.



Real Time Devices USA

2100 Innovation Blvd. • P.O. Box 906
State College, PA 16803 USA
(814) 234-8087 / Fax: (814) 234-5211

R TD Europa • RTD Scandinavia

Real Time Devices is a founder of the PC/104 Consortium.

```
ap to row 0
Firmware Furnace Task Switcher -- Ed Nisley (c) 1994 CAJ
Issue 53: Video and LCD support

Colors...
ON GREEN... ON BLUE
Yellow on blue again

Cursor positioning...
0123456789012345678901234567890123456789012345678901234567890
Col 10 Col 40 Col 65

Tab stops...
0123456789012345678901234567890123456789012345678901234567890
This is tabbed with... wrap at end

12348AE3

ap to row 24
Row 0 Col 65
Row 1 Col 65
Row 23 wr
Row 24 wr
```

Photo 1--This test pattern exercises most of the video driver's functions by displaying text in a fixed format. Because the driver does not scroll, the screen text wraps from the lower-right corner to the upper-left. The eight-character counter just below the tab-stop test shows an incrementing value that may be slightly blurred due to photo exposure. You can't see blinking text on this page, either, sad to say.

column, 25-line display requires 4000 (decimal) bytes (don't forget the attributes!) and thus has a 4096-byte page size. Even though those 96 bytes at the end are not visible on the screen, I felt tweaking the official BIOS page size wasn't worthwhile.

As with all protected-mode segments, any write beyond the video page will trigger a protection exception. If you need access to other pages, you must either expand the segment or create additional descriptors for the new pages. Using one descriptor per page is a nice way to handle output from separate tasks.

In fact, putting the screen-page descriptor in the task's LDT ensures that it cannot write into any other page. You can have several tasks, each using the same LDT selector to access the screen, but each selector refers to a different chunk of the video buffer. Might come in handy, indeed!

Oddly enough, the BIOS doesn't store the video-buffer address in its data area. It does, however, save the CRT controller I/O port address at 0040:0063, and that gives us enough to locate the buffer. The bad news is the BIOS stores the MDA address, even if no video board is installed.

Listing 3 presents the video-initialization code. It extracts several values from the BIOS data area, creates the segment descriptor covering the buffer, then performs a simple memory test. If the buffer holds data correctly, the code sets a status flag that enables the remaining video functions, turns off the cursor, and clears the screen.

The MemPeekReal() function converts a real-mode segment:off address into a 32-bit linear address and returns the double word at that address in EAX. The familiar shift-segment-left-four and add-the-offset dance produces a 20-bit value. The protected-mode startup code created a read-only descriptor that maps all of installed memory starting at address 00000000. That 20-bit address is our first non-trivial 32-bit flat address, albeit with a dozen high-order zeros.

MemSetDescriptor() handles all the hocus pocus required to load a descriptor entry. The code is similar to Listing 6c in last month's column. The functions accessing the buffer load a 48-bit full pointer called CurPointer using LES instructions. CurPointer's segment component is just the GDT_VIDEO selector for the video-buffer descriptor.

The proof of the coding is in the viewing. Photo 1 shows the video test pattern you should see when you run this month's code. The first few words on the top line are wrapped from the bottom to demonstrate that the screen does not scroll vertically. The driver supports cursor positioning, color changes, and the usual CR, LF, and tab control characters.

The eight-hexit number a few lines below the tab-stop test pattern is a double-word counter driving a simple binary-to-ASCII conversion routine. The last digit or two may be slightly blurred because of the photo's lengthy exposure time. You'll also see a moving dot on the LPT1LEDs to indicate that the test code is really alive inside your system.

That's enough to let FFTS report back in style!

DOING IT WITH DOTS...

I originally planned to use the FDB's Graphic LCD Interface for this part of the project. The folks at the local robotics club convinced me that standard VGAs outnumbered hand-wired LCD panels by umpty-zillion to one. My wounds are healing nicely, thank you very much.

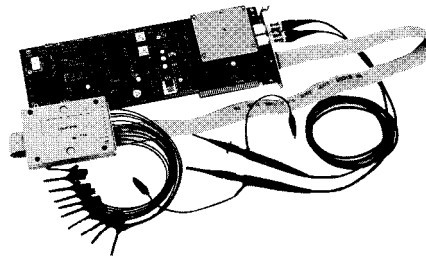
As it turns out though, the top-level code is essentially identical for both displays. I suppose a device-independent interface with automatic color mapping would be a nice touch, but you'll probably just omit the code for the hardware you don't have.. and that simplifies things a lot.

Earlier this year, you saw that graphic LCD panels have a wide variety of interfaces, timing specifications, and dot arrangements. The low-level, hardware-dependent code required for each panel is bottled up in an assembler file with an obvious name: DMF651 . ASM, for instance. While the drivers may work with similar panels, you must verify that on your own.

Refer back to CAJ 46 for a cram course on the BIOS CGA font table and similar matters since I've recycled some of that code into 3%bit protected-mode assembler. I'll skip the detailed background discussions here to save space.

PC-Based Instruments 200 MSa/s DIGITAL OSCILLOSCOPE

**HUGE BUFFER
FAST SAMPLING
SCOPE AND LOGIC ANALYZER
C LIBRARY W/SOURCE AVAILABLE
POWERFUL FRONT PANEL SOFTWARE**



\$1799 - DSO-28204 (4K)

\$2285 - DSO-28264 (64K)

DSO Channels

2 Ch. up to 100 MSa/s

or

1 Ch. at 200MSa/s

4K or 64K Samples/Ch

Cross Trigger with LA

125 MHz Bandwidth

Logic Analyzer Channels

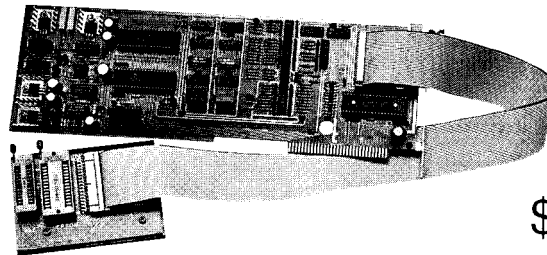
8 Ch. up to 100 MHz

4K or 64K Samples/Ch

Cross Trigger with DSO

Universal Device Programmer

**PAL
GAL
EPROM
EEPROM
FLASH
MICRO
PIC
etc..**

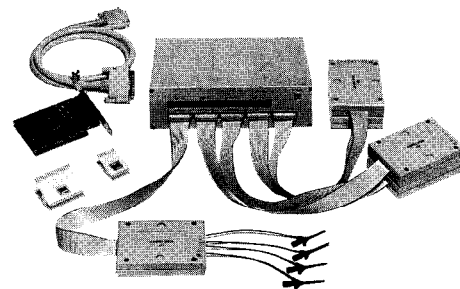


\$475

Free software updates on BBS
Powerful menu driven software

400 MHz Logic Analyzer

- up to 128 Channels
- up to 400 MHz
- up to 16K Samples/Channel
- Variable Threshold Levels
- 8 External Clocks
- 16 Level Triggering
- Pattern Generator Option



\$799- LA12100 (100 MHz, 24 Ch)

\$1299- LA32200 (200 MHz, 32 Ch)

\$1899- LA32400 (400 MHz, 32 Ch)

\$2750- LA64400 (400 MHz, 64 Ch)

Call (201) 808-8990

Link Instruments

369 Passaic Ave, Suite 100, Fairfield, NJ 07004 fax: 808-8786

The LCDWriteGlyph routine in Listing 4 converts an ASCII character into the appropriate bit pattern from the BIOS 8 x 8 CGA font. The code substitutes a question mark for any characters beyond the 128 present in that table. Multiplying the resulting numeric character value by the font height gives the character's offset from the start of the BIOS table.

LCDWriteGlyph also combines the current cursor location with the font height and width to get the dot address of the upper-left corner of the character cell on the panel. This calculation is easy because the characters are always aligned on an 8 x 8-dot grid. If you want proportional fonts (yikes!), this is the place to keep track of character widths on each line.. which I leave as an exercise.

The compatibility barnacles anchor the CGA font table to the same address in every PC. That makes the 48-bit c p Font pointer a simple constant in the _p r o t . c o n s t segment. You can substitute a different fixed-pitch font table by aiming c p F o n t at it and tweaking the font-size constants. If you have a VGA card, you can filch its bitmapped fonts to trade off information density for eye appeal using the techniques I covered in CAJ 46.

The LCDWriteGlyph loop fetches successive bytes from the font table and writes them into the refresh buffer using the low-level LCDWriteByte routine. Each panel has a different dot layout, which means a custom routine must distribute the dots into the buffer. That code translates blinking characters into something useful on panels that don't support blinking.

For more grubby, bit-twiddling details, check the source code on the BBS. I've written drivers for three different panels in the hopes they'll either match what you have or be close enough that you can adapt the code without too much trouble.

The test pattern on a TLY365 is essentially the same as Photo 1. The last few digits of the double-word counter are harder to read because LCD panels have a slower response time. If you don't have the LCD interface installed, the code will simply disable itself.

Listing 2-Choosing a simple binary encoding makes interpreting the cursor and color-control commands a/most trivial. The loop in Listing 1 defects theVIDCMD byte preceding each command and invokes this routine to examine the remaining bytes. The next byte determines the operation and specifies how many data bytes are included. If you add more commands, a table-driven decoder would tidy up the code by eliminating the chain ofCMPs.

```

PROC    Vi dCommand
USES    EAX, EBX

        LODS    [BYTE PTR ES:ESI]    ; pick up command byte

        --- cursor positioning

        CMP     AL, VIDROW
        JNE     SHORT @@NotRow

        LODS    [BYTE PTR ES:ESI]    ; set new row
        AND     EAX, 07Fh
        CALL    VidSetRowCol, EAX, [CurCol]
        JMP     @@Done

@@NotRow:
        CMP     AL, VIDCOL
        JNE     SHORT @@NotCol

        LODS    [BYTE PTR ES:ESI]    ; set new column
        AND     EAX, 07Fh
        CALL    VidSetRowCol, [CurRow], EAX
        JMP     SHORT @@Done

@@NotCol:
        <<< code to set both row & column omitted >>>
        JMP     SHORT @Done

@@NotRC:
        ; --- on-the-fly color changes

        <<< code to set foreground & background omitted >>>

        CMP     AL, VIDFGBG
        JNE     SHORT @@NotFGBG

        LODS    [BYTE PTR ES:ESI]    ; set foreground
        AND     EAX, 0000000Fh
        MOV     EBX, EAX
        LODS    [BYTE PTR ES:ESI]    ; and background
        AND     EAX, 0000000Fh
        SHL     EAX, 4
        OR      EAX, EBX
        MOV     [CurAttr], EAX
        JMP     SHORT @@Done

@@NotFGBG:
        NOP

@@Done:
        RET

ENDP    Vi dCommand

```

PM PERFORMANCE

The video-output routines are the first nontrivial 32-bit protected-mode code we've seen so far, although I'd argue that just getting into 32-bit PM is nontrivial enough for most purposes. In any event, the question comes up: how fast does this stuff run, anyway!

The counter shown on each display provides a convenient way to collect some data because the code is running in a known pattern with all interrupts disabled. I flipped the LPT1 strobe bit at key points during the test loop to produce the upper trace in Photo 2.

Listing 3—*The protected-mode descriptor covering the video buffer must include its starting address and length. The BIOS initializes several key values in its data area during each reset, making it reasonably easy to figure out what kind of video hardware is installed. The MemPeekReal() function returns the double word at the protected-mode address corresponding to a real-mode segment:off address.*

```

PROC   VidInitialize
USES   EAX,EBX,ESI,ES

    extract a few BIOS variables for our use

    CALL MemPeekReal,BIOS_SEG,4Ah; columns
    MOVZX EAX,AX
    MOV   [NumCols],EAX

    CALL MemPeekReal,BIOS_SEG,84h; rows 1
    MOVZX EAX,AL
    INC   EAX
    MOV   [NumRows],EAX

    CALL MemPeekReal,BIOS_SEG,63h; CRT controller addr
    MOVZX EAX,AX
    MOV   [CRTCBase],EAX

    ADD   EAX,06h                ; status port address
    MOV   [CRTStatus],EAX

;--- create a GDT descriptor covering the video buffer
;we tweak the memory starting address based on the CRTC I/O
;address...and, if it's a color display, we assume it's a VGA

    CALL MemPeekReal,BIOS_SEG,4Ch; video page length
    MOVZX EBX,AX
    MOV   [PageLength],EBX

    MOV   EAX,000B8000h          ; assume color
    MOV   [CRTAttrCtl],03c0h     ; for VGA Attribute Ctl
    CMP   [CRTCBase],03D4h
    JE    @@UseColor
    MOV   EAX,000B0000h          ; assume monochrome
    MOV   [CRTAttrCtl],0         ; disable this access
@@UseColor:

    CALL MemSetDescriptor,GDT_VIDEO,GDT_GDT_ALIAS, \
    EAX,[PageLength],ACC_DATA32,ATTR_32BIT

;--- see if the video buffer actually holds data
;if not, disable the video functions

    MOV   [CurPointer.Seg],GDT_VIDEO; set char pointer

    LES   ESI,[FWORD PTR CurPointer]
    MOV   AL,[ES:ESI]            ; fetch it
    MOV   AH,AL                  ; save for later
    NOT   AL                     ; flip all the bits
    MOV   [ES:ESI],AL            ; write it out
    CMP   AL,[ES:ESI]            ; see if it stuck
    MOV   [ES:ESI],AH            ; restore original value
    JNE   SHORT @@NoVideo        ; skip if no match
    INC   [VideoEnabled]         ; indicate that we're OK
@@NoVideo:

;--- these functions will bail out if video is disabled

    CALL VidTurnCursorOff
    CALL VidClearScreen,VID_DEFAULT

    RET

ENDP   VidInitialize

```

The VGA, running in text mode, writes eight pairs of attribute and character bytes in 270 μ s, or 34 μ s per character. I eyeballed the code listing to come up with about 370 instructions for the complete display. That works out to about 1.4 MIPS or, inversely, 730 ns per instruction. The '386SX CPU is running at 33 MHz, implying that each instruction requires 24 clock cycles.

The Graphic LCD Interface is a bitmapped graphics device with a byte-wide data path. Each character requires eight font-table reads and, for the TLY365, 16 writes into the refresh buffer. The lower trace in Photo 2 shows those accesses blotting up a total of 2 μ s, or 250 μ s per char. Another eyeball count reports 2700 instructions or 1.4 MIPS again.

Bear in mind that those averages include the '386SX bus-interface overhead for 32-bit data and stack accesses through a 16-bit data path, ISA bus delays, prefetch queue flushes, DRAM refresh interference, and memory wait states. The instruction-cycle counts that you read in the manuals quietly exclude all that, giving the novice a rather optimistic view of the world.

Homework assignment: if you think this is lots worse than real mode, recode the test program to run in 16-bit real mode, make the same measurements, and report back on the BBS. My guess is that real mode will be about 10–15% faster—maybe 20 clock cycles per instruction instead of 24. Hmmm?

This code has an unusually high number of accesses to unusually slow memory. The system-board memory runs much faster than the video and LCD RAMs on the ISA bus. For extra credit on your homework: map the accesses into fake buffers in system memory and retime the code. I bet you'll pick up another 10% right there!

THE CASE OF THE CAPITAL "T"

When I wrote the TLY365 driver, my '386SX developed a curious problem. The LCD panel worked fine, but the system hung midway through the next BIOS boot sequence after I pressed the reset button. Cycling the

power worked fine. It hung reliably after every manual reset. Hmmmm...

A little probing showed that the CPU was stuck in a loop doing a little I/O and a lot of memory writes. It surely wasn't any of my code because I didn't have any BIOS extensions installed. Just to make sure, I pulled the battery-backed RAM out of the Firmware Development Board's socket. The CPU still got wedged.

For lack of a better idea, I pulled the Graphic LCD Interface's refresh RAM. As you might expect, the system worked perfectly even though the LCD wasn't displaying much of anything. Swapping RAM chips didn't solve the problem.

The system worked correctly with the DMF65 1 and LG64AA44D panels and the appropriate test code. The TLY365 worked OK with the Game of Life and ANSI test code from earlier this year. It failed only after displaying the test pattern in protected mode.

I modified the test pattern to write all blanks into the panel's RAM, and found that the system boot normally. A quick divide-and-conquer search showed the failure occurred when a "T" appeared in the first position of line 12. No other characters seemed to matter: "Tab stops.. ." failed just like a single "T" followed by blanks.

I whipped out my jeweler's loupe, examined the panel, then drew up Figure 1 showing the bit patterns, dot row numbers, and RAM addresses for character line 12. If you've been paying attention for the last year or so, the problem should be obvious.

This is a quiz!

OK, here's the story. Twelve lines (0-11) of 8 x 8 BIOS character cells puts the top bar of the T on dot row 96. Each dot row occupies 320 bytes of RAM because the TLY365 has 1280 dots on each of 100 rows, arranged in a 640 x 200 physical array. Row 1 starts at address 0000 (for reasons I covered in CAJ43), which puts row 96 at address $(96-1) \times 320 = 76C0$.

Row 97, the second row of the character cell, begins at address 7800. The dot pattern for that row begins with the tips of the T's serifs and its two-dot-wide central stroke. The hex value is 5A, which looks suspicious

Listing 4—The Graphic LCD Interface does not include a hardware character generator. This routine draws character glyphs into the LCD refresh buffer using the BIOS CGA font table. Segment register FS contains the GDT_CONST selector needed to read values stored in the _p r o t c o n s t segment. Their names start with a lowercase "c" to indicate that they are not in the same segment as the usual read-write variables.

```

PROC    GLCDWriteGlyph
ARG     CharValue: DWORD, CharRow: DWORD, CharCol: DWORD, \
        Attribute: DWORD
LOCAL   DotRow: DWORD, DotCol: DWORD
USES    EAX,EBX,ECX,ESI,ES

MOVZX   EAX,[BYTE PTR CharValue]    ; can we draw it?
CMP     EAX,[FS:cNumFontChars]
JB      SHORT @@CharOK
MOV     AL,'?'                       ; nope, flag the char
@@CharOK:

IMUL    [BYTE PTR FS:cFontHeight]    ; get char offset in font
LES     ESI,[FWORD PTR FS:cpFont]    ; pick up font base
ADD     ESI,EAX                      ; add char offset
MOV     ECX,[FS:cFontHeight]         ; set up row counter

MOVZX   EAX,[BYTE PTR CharRow]        convert coordinates
IMUL    [BYTE PTR FS:cFontHeight]    into dots
MOV     EBX,EAX
MOVZX   EAX,[BYTE PTR CharCol]
IMUL    [BYTE PTR FS:cFontWidth]
MOV     EDX,EAX
XOR     EAX,EAX                      preload OFF dots

@@NextRow:
MOV     AL,[ES:ESI]                  ; pick up dot row
CALL    LCDWriteByte,EAX,EBX,EDX,[Attribute]
INC     ESI                          ; next font line
INC     EBX                          ; and next screen line
LOOP    @@NextRow

RET

ENDP    GLCDWriteGlyph

```

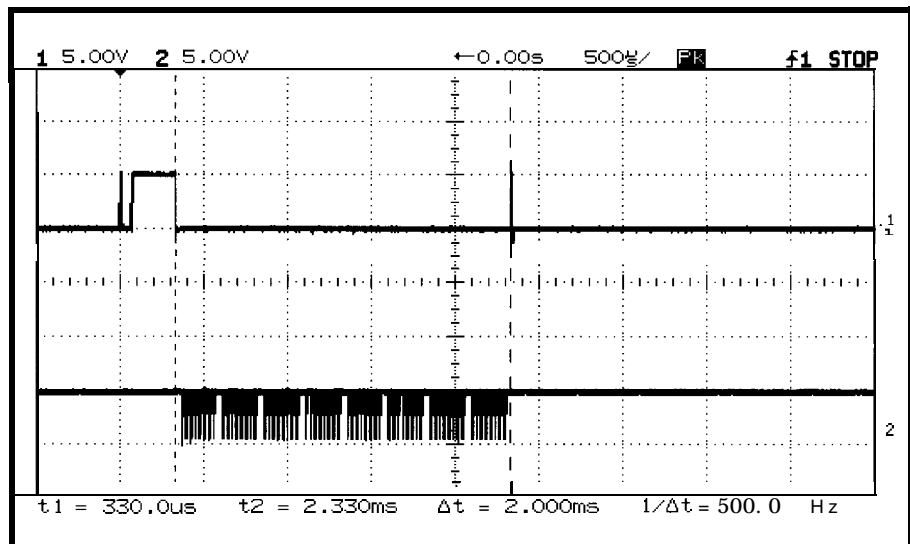


Photo 2—This scope shot shows how rapidly the VGA and TLY365 LCD drivers write the eight-character counter value to their respective displays. The VGA driver is active during the second pulse on the upper trace; it takes about 35 µs per char. The LCD driver starts immediately after that. Drawing each row of each character using the BIOS font table requires about 250 µs per char. Each of the eight-pulse groups in the lower trace correspond to one character. Each character has 8 rows, and each row requires 2 LCD refresh buffer writes.

already. Recall that the TLY365 has a four-bit data interface and the Graphic LCD Interface implements blinking by alternating between the two nybbles of each refresh RAM byte.

Because the T isn't blinking, the refresh-buffer bytes at addresses 7800 and 7801 have identical nybbles: 55 AA. If that doesn't raise your hackles, you flunk..

Recall that the BIOS boot routine scans memory between C000 and EFFF for BIOS extensions. By definition, a BIOS extension must start on a 2-KB memory boundary with two flag bytes and the length of the extension in multiples of 512 bytes. Yes, the flag bytes are 55 and AA.

The second character on line 12 was either a lowercase "a" or a blank, neither of which has any dots on row 97. That translates into a pair of binary zeroes in the refresh RAM after the 55 and AA.

The test pattern's first three bytes define a BIOS extension starting at 7800 with a length of zero bytes.

Gotcha!

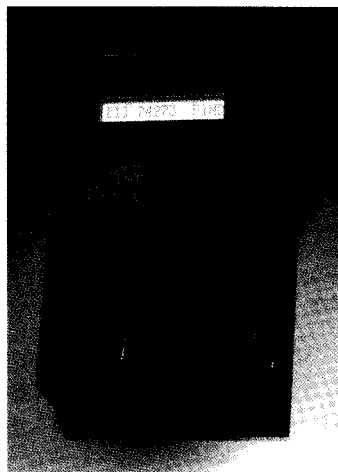
A valid BIOS extension also includes a checksum byte to make the sum of all the bytes defined by the length equal to zero. Evidently, the BIOS checksum routine in my PC concludes that a zero-length extension, lacking any content, is always valid. Remember that the refresh buffer contents after the header had no effect on whether the CPU got wedged.

So the situation goes a little something like this..

During a power-on reset the BIOS finds nothing particular in the LCD refresh RAM and boots normally. My protected-mode code displays a test pattern on the LCD panel which, quite accidentally, plunks what looks like a BIOS extension header on a 2-KB boundary within the refresh buffer.

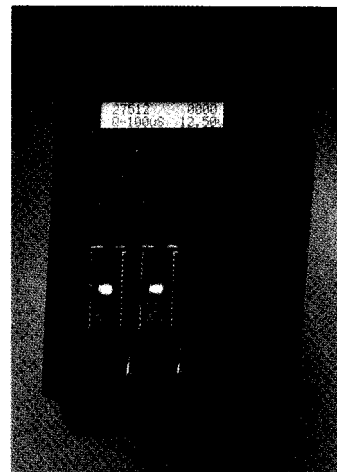
Pressing reset sends the BIOS through its extension scan again, where it finds the header at address D000: 7800. It (erroneously) concludes that the extension's checksum is valid and branches to offset 7803 in the LCD refresh buffer, which is the second zero byte. What happens after that is up for grabs. On this system, the CPU finds a loop that never ends.

Data Genie offers a full line of test & measurement equipment that's innovative, reliable and very affordable. The "Express Series" of stand-alone, non-PC based testers are the ultimate in portability when running from either battery or AC power. Data Genie products will be setting the standards for quality on the bench or in the field for years to come.



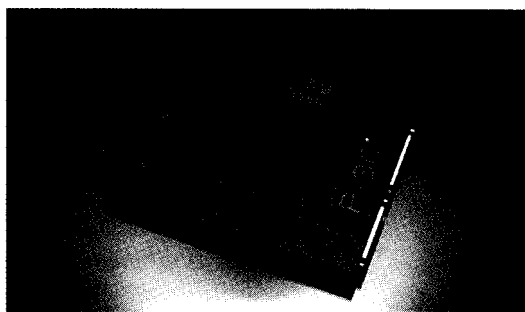
HT-28 Express

The HT-28 is a very convenient way of testing Logic IC's and DRAM's. Tests most TTL 74, CMOS 40/45 and DRAM's 4164-414000, 44164-441000. It can also identify unknown IC numbers on TTL 74 and CMOS 40/45 series with the 'Auto-Search' feature.
\$189.95



HT- 14 Express

The HT-14 is one-to-one EPROM writer with a super fast programming speed that supports devices from 27328 to 27080, with eight selectable programming algorithms and six programming power (VPP) selections.
\$289.95



P-300

The Data Genie P-300 is a useful device that allows you to quickly install add-on cards or to test prototype circuits for your PC externally. Without having to turn off your computer to install an add-on cards, the P-300 maintains complete protection for your motherboard via the built-in current limit fuses.
\$349.95

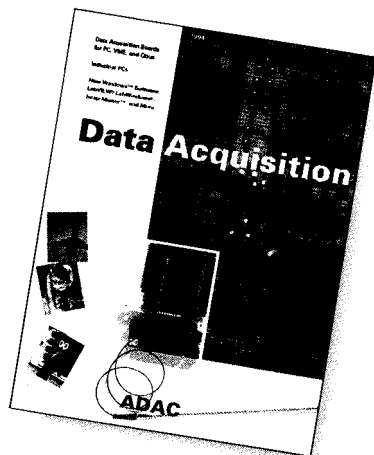
MING
Microsystems
Division of MING & P. INC.
17921 Rowland Street
City of Industry, CA 91748
TEL: (818) 912-7756
FAX: (8113) 912-9598

Call for a dealer near you.
1-800-473-6606

Data Genie products are backed by a full year limited factory warranty

NEW Data Acquisition Catalog

Covers expanded low cost line.



FREE!

NEW 120 page catalog for PC, VME, and Qbus data acquisition. Plus informative application notes regarding anti-alias filtering, signal conditioning, and more.

NEW Software:
LabVIEW®, **LabWindows®**,
Snap-Master™, and more

NEW Low Cost I/O Boards

NEW Industrial PCs

NEW Isolated Analog and Digital Industrial I/O

New from the inventors of plug-in data acquisition.

Call, fax, or mail for your free copy today.

ADAC

American Data Acquisition Corporation
70 Tower Office Park, Woburn, MA 01801
Phone: (800) 648-6589 Fax: (617) 938-6553

#122

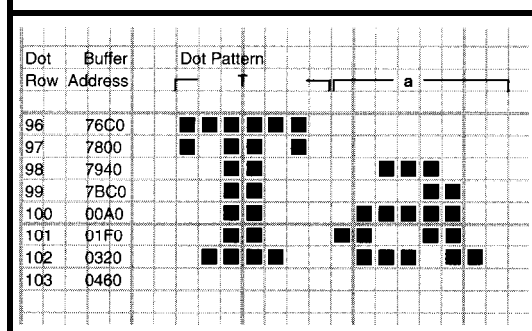


Figure 1--The LCD test code writes "Tab stops." on Line 12. This figure shows the LCD refresh buffer addresses and bit patterns corresponding to the first two letters for the TLY365 640 x 200 panel, which is electrically a 1280 x 100 panel. The Graphic LCD Interface hardware blinks by alternating the upper and lower nybbles of each byte, so every four dots on the panel require an 8-bit byte in the buffer.

So much for real mode, hmmm!
The solution is easy enough:
disable the LCD refresh RAM when the system reset line goes active so the BIOS scan cannot find a bogus extension in the buffer. The Firmware Development Board sprouted a MAX691 watchdog timer in CAT 37, with a latch to stretch the timeout interval after a reset. That extra guardian hardware provides the signal we need to keep the BIOS under control.

Add a wire from the -Q output of the latch (U18.8) to the RAM's -CE input (U45.20). The FFTS code sets that latch and enables the RAM chip shortly after the CPU enters protected mode. The LCD driver will activate the 8254 timer, clear the buffer, and set up the test pattern fast enough that you'll see just a blink of the previous buffer contents.

A different cure would be a (deliberate!) BIOS extension in the FDB's battery-backed RAM that gets control before the BIOS hits the refresh buffer. That extension would set up the Graphic LCD Interface for the particular panel and clear the buffer to ensure the BIOS doesn't find anything disturbing. If you've converted PMLoader to an extension, just add the requisite lines of code.

This error is an oversight, pure and simple. I knew (and so did you!) that the LCD refresh RAM contents survived a reset. It never occurred to me that the BIOS might discover an extension in the bit patterns left over from the last display!

If you build anything with dynamic bit patterns in the region where the BIOS expects extensions, take heed. You, too, may spend hard time wondering why your system doesn't boot correctly once in a while.

In a few columns, the watchdog timer will stand guard over the FFTS kernel, making that port output part of the normal startup activities. Until then, just watch the watchdog LED blink merrily along at its fast rate.

RELEASE NOTES

The code this month displays a test pattern on both a 640 x 200 Toshiba TLY365 LCD panel and an ordinary VGA. The drivers for 640 x 200 Optrex DMF65 1 and 640 x 400 Matsushita LG64AA44D panels are included; just uncomment the appropriate line in MAKE F I L E and rebuild FFTS. PM0 to suit your system.

The test pattern still includes that fateful T, so you should add the wire to disable the RAM after each system reset. I suspect many BIOSs ignore zero-length extensions and reject invalid checksums. Don't take any chances. After all, a bit pattern that looks like a valid extension will occur just before your big demo..

The video driver should also work with a text-only monochrome adapter or an old Hercules card, but I can't test that here. If you have the appropriate hardware, give it a shot and report back on the BBS.

Next month we start '386SX multitasking and measure just how long a task switch really takes. □

Ed Nisley, as Nisley Micro Engineering, makes small computers do amazing things. He's also a member of the Computer Applications Journal's engineering staff. You may reach him at ed.nisley@circellar.com or 74065.1363@compuserve.com.

I R S

413 Very Useful
414 Moderately Useful
415 Not Useful