# 13 Bitmapped LCD Panel Hardware

Perhaps we've lost our sense of wonder: name another manufactured product with a quarter of a million parts laid out for your inspection. Every dot on that 640×400 LCD panel must be essentially perfect, because human eyes are very, very good at picking out tiny differences. Sure, a CD-ROM has more dots, but it's just a silver disk to anything but an electron microscope.

In this chapter, we will build the hardware that puts every one of those dots under your control. I'll concentrate on the details of 200- and 400-line displays, because they're readily available, while mentioning some of the changes required for other panels. The block diagram in the previous chapter will guide you through the circuitry in the schematics you'll find here.

Wading through the intricate timing relationships of the sync and data signals takes considerable mental effort. This is, alas, typical of most hardware projects, with the Devil hiding in the details. Getting even one detail wrong can wreck a project, so pay attention as we go along and see if I've missed anything!

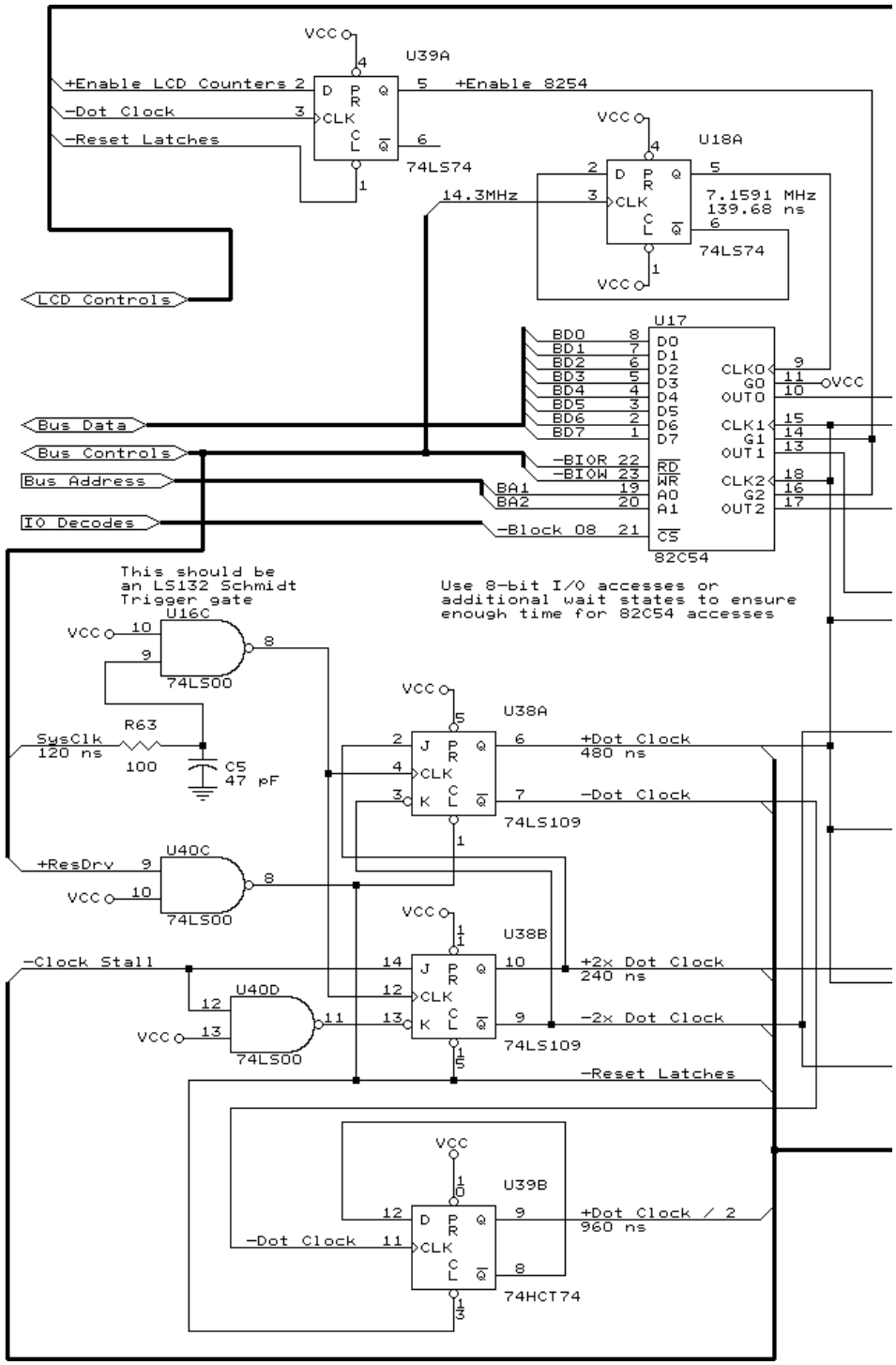
You should peek ahead to Chapter 14, where I cover the test code that checks out the circuitry, before you fire up your soldering iron. By verifying your work in small steps, you'll save yourself lots of effort… I know I did.

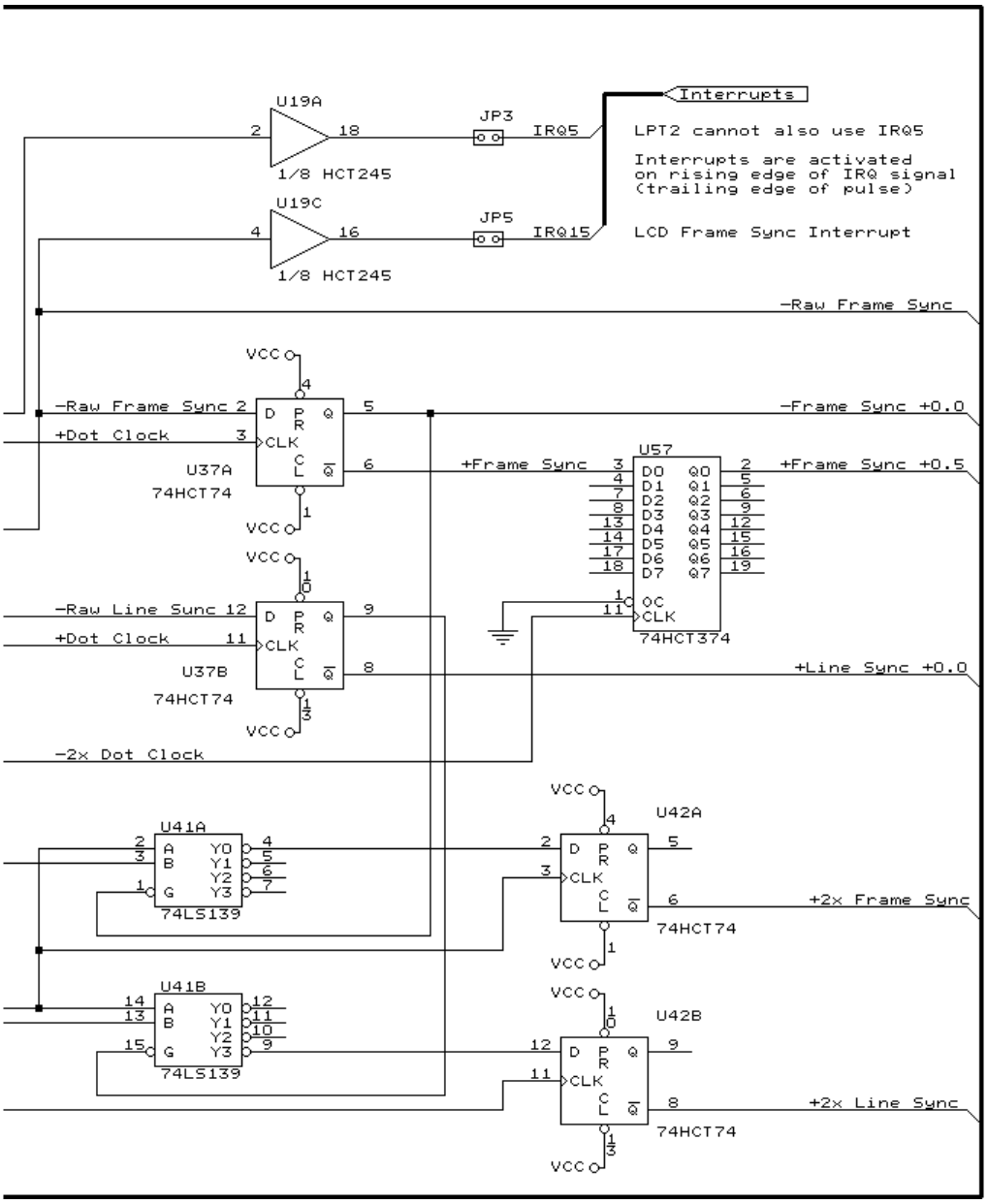
## Counting Clocks

The LCD Timing and Control section forms the heart of the Graphic LCD Interface. It generates the clock and sync pulses that control the LCD panel, as well as the signals that orchestrate the PC side of the interface. Schematic 1 shows this part of the circuit.

The ISA bus **SysClk** signal defines the basic interface timing. U38, an LS109 dual J/-K flipflop, divides the 120 ns **SysClk** by two to produce the double-speed **2x Dot Clock**, then by two again for the regular 480 ns **Dot Clock**. U39B, part of an HCT74, reduces that rate for the half-speed 960 ns **Dot Clock/2** required by slower panels.

The actual **SysClk** frequency, and thus the fundamental ISA bus cycle, for any given system may range from about 120 to 125 ns, because the system logic generates it from the CPU clock. As a rule of thumb, the bus cycle will be a simple integer multiple of the fundamental clock cycle, although recent systems include phase-locked loops with fractional divisors.


VCC
U39A
+Enable LCD Counters
-Dot Clock
-Reset Latches
+Enable 8254
74LS74
U18A
14.3MHz
7.1591 MHz
139.68 ns
74LS74
LCD Controls
U17
BD0
BD1
BD2
BD3
BD4
BD5
BD6
BD7
Bus Data
Bus Controls
Bus Address
IO Decodes
-BIOR
-BIOW
BA1
BA2
-Block 08
82C54
This should be an LS132 Schmidt Trigger gate
Use 8-bit I/O accesses or additional wait states to ensure enough time for 82C54 accesses
U16C
74LS00
SysClk
120 ns
R63
100
C5
47 pF
U38A
+Dot Clock
480 ns
-Dot Clock
74LS109
U40C
+ResDrv
U38B
-Clock Stall
U40D
+2x Dot Clock
240 ns
-2x Dot Clock
-Reset Latches
U39B
+Dot Clock / 2
960 ns
74HCT74




Schematic 1

This circuitry defines the fundamental Dot Clock period that drives the rest of the Graphic LCD Interface. The 82C54, installed in Chapter 4, has new clock and gate lines. The various Dot Clock and Sync signal outputs can control a variety of panels.

For example, an 80 MHz 80486DX2 system uses a 40 MHz external clock that, divided by 5, produces an 8.000 MHz bus clock and a 125 ns cycle time. An old 33 MHz 80386SX, which actually uses a simple 33 MHz clock, has an 8.25 MHz bus and a 121 ns **SysClk** cycle. There's some specsmanship involved with these numbers, so don't be surprised if your **SysClk** seems a bit different than you expect. I'll continue to use 120 ns here, as it makes the numbers work out evenly.

In Chapter 4, when we first wired up the Firmware Development Board's 82C54 timer, we tied all of the **Gate** inputs high and drove the **Clock** inputs from a 7.16 MHz signal. Now, we can put that hardware to practical use! U17 in Schematic 1 represents the same 82C54 you've already checked out, with two channels producing **Line Sync** and **Frame Sync** signals for our panels.

Counter 1 produces the **-Raw Line Sync** signal by counting the number of **Dot Clock**s in each LCD row. The 640×200 and 640×400 panels I have require either 160 or 320 clocks per row, depending on whether they use a single- or double-speed interface. Because other panels have different requirements, using a programmable counter made more sense than a tangle of jumpers and gates.

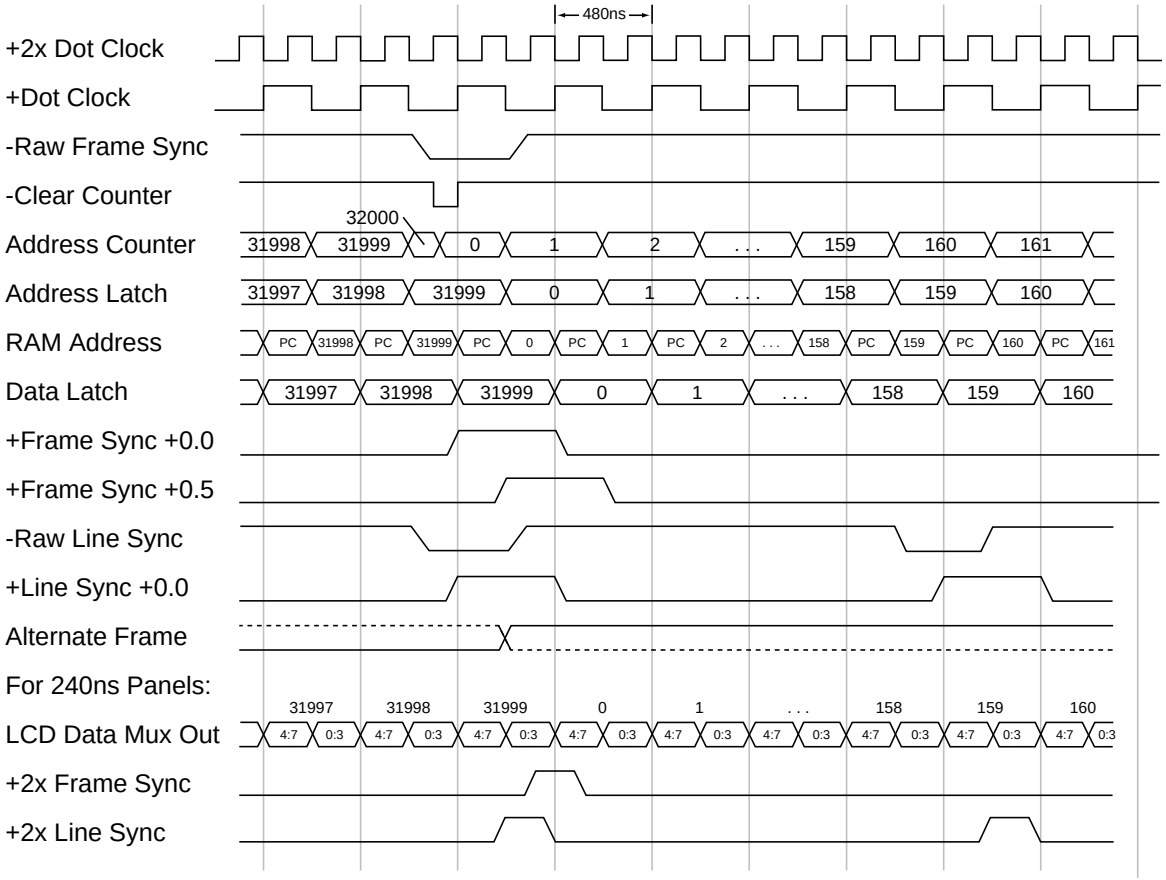
Counter 2 controls the number of **Dot Clock** cycles in one LCD display frame by generating the **-Raw Frame Sync** signal. Both 200- and 400-line panels require 32000 clocks, but other panels have different counts. Here, also, the 82C54 serves as a compact, 16-bit, programmable counter.

The key Graphic LCD Interface timing signals appear in Figure 1. The 82C54 timing specs say the output delay may occur up to 100 ns after the falling **Clock** input edge, which means you can't use the signals for critical timings. I used an HCT74 to synchronize the edges with **+Dot Clock**; the resulting pulses occur at the "**+0.0**" times relative to the rest of the signals.

Both 82C54 channels must begin counting on the same **Dot Clock** cycle to ensure that the two sync signals have the proper relationship. U39A disables the counters until the firmware sets the **+Enable LCD Counters** bit in the **LCD Control Port**. When U39A's clock rises, both 82C54 **Gate** inputs go high simultaneously and the counters are off and running.

While I was debugging this circuitry, the LCD panel would occasionally go blank. It turned out that Counter 1 unpredictably stopped counting for intervals ranging from a few cycles to a few hundred milliseconds. This immediately threw off the critical timing relationship between **Line Sync** and **Frame Sync**. Strangely enough, the 82C54 **Gate** inputs remained high and **Dot Clock** continued to run during the stoppages.

Figure 1
This timing diagram summarizes the Graphic LCD Interface timings. The 32 KB RAM holding the LCD dot patterns is timeshared between the PC ISA bus and the LCD circuitry. The -Clear Counter signal resets the LCD Address Counter at the end of each frame of data; that signal originates in the 82C54 counter described in Chapter 4.



The 82C54 has a classic "read every word" data sheet. The **Clock** input timing specs include this tiny footnote: "Low-going glitches that violate $t_{PWH}$, $t_{PWL}$ may cause errors requiring counter programming." Translated into plain English, this says that an imperfect **Clock** will cause the counter to misbehave in strange and mysterious ways. Hence, the RC filter and gate driving U38.

We think of TTL devices as digital: their outputs will be either high or low, never anywhere in between. In reality, of course, digital logic circuits are just analog circuits with a nonlinear transfer characteristic. If the input voltage falls between the valid logic 1 and logic 0 thresholds, you will certainly get *something* out… perhaps not a Boolean value and usually nothing that you really want.

Further investigation revealed **Dot Clock** was something other than the picture-perfect square wave I expected. Once in a while, a low-going glitch dented the high half cycle. After some delicate probing, I discovered an impressively noisy **SysClk** signal on the ISA bus. The clock noise sometimes became bad enough to force the LS109 latches into metastable states. The ensuing invalid output voltages stunned one or both of the 82C54 counters.

The effect was that the counters simply stopped counting for a while, then started up again after an unpredictable interval. Of course, when **Frame Sync** and **Line Sync** pulses don't occur in the proper relationship, the panel doesn't display anything useful. Although scope probes on the sync signals show that they're cooking right along, you can't easily see the phase relationship on a scope and may miss some crucial evidence.

Because we cannot clean up the ISA bus logic signals, we must cope with what arrives at the Firmware Development Board. One possible solution for this problem puts a Schmidt Trigger gate in the clock path to eliminate the noise. Unlike a standard logic gate, a Schmidt Trigger includes an analog feedback path that modifies the input switching thresholds based on the Boolean output. As a result, small glitches around the thresholds simply Go Away. However, I didn't want to squeeze an LS14 on the board to get just one inverter.

So, I used a simple RC filter to knock off the high-frequency noise, followed by a spare LS00 gate to square up the result. Yes, I could have substituted an LS132 (a Schmidt Trigger NAND), but the *other* circuits already using that package couldn't stand the additional delay. An RC filter on a (nominally) digital input, albeit not a textbook solution, works for this application. Take a look at *your* **SysClk**, consider all your options, then decide how you will attack the problem.

You may rest assured that it took me *much* longer to discover and stomp that glitch than to write about it…

Although the 82C54 defines the overall row and frame timing, its output pulses cannot drive the LCD panel directly. The dot data and sync signals must all arrive with the correct polarities at the proper times. Now, let's consider the dots.

## A Chain of Dots

A 32 KB static RAM holds the binary data that will become visible dots on the panel. The **LCD Refresh RAM**, U45 in Schematic 2, must timeshare its signals between the ISA bus and the LCD panel. Obviously, only the PC will write into the RAM, but both sides must be able to read the dot data. The **Dot Clock** signal determines which side has access to the RAM at any given moment.

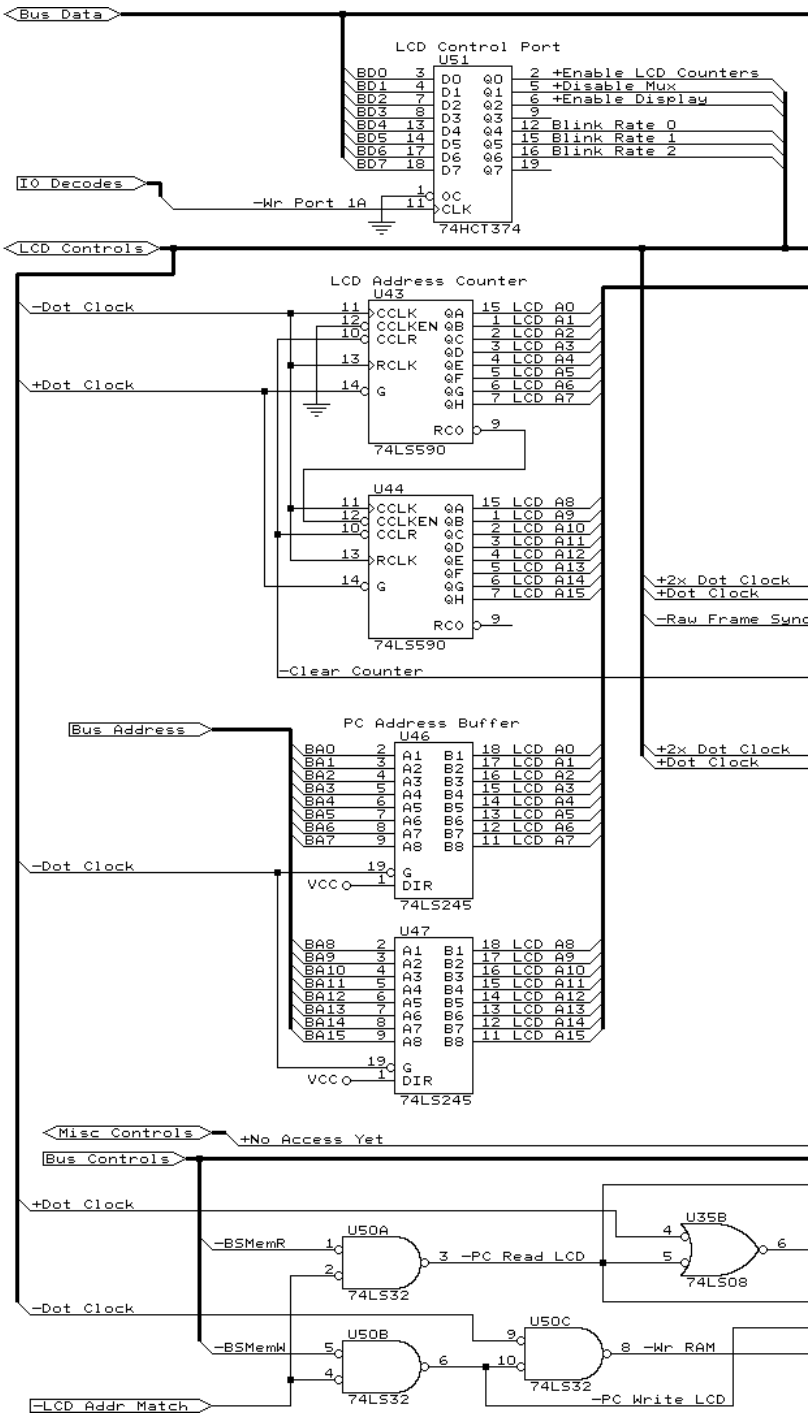
The **LCD Address Counters**, U43 and U44, provide the 15-bit address of the current RAM byte when the LCD has access. Each LS590 chip contains an 8-bit counter followed by an 8-bit latch with three-state outputs. Both the counter and latch change state on the rising edge of the clock input, which, in this circuit, is **-Dot Clock**. Therefore, the LCD address changes on the *falling* edge of **+Dot Clock**.

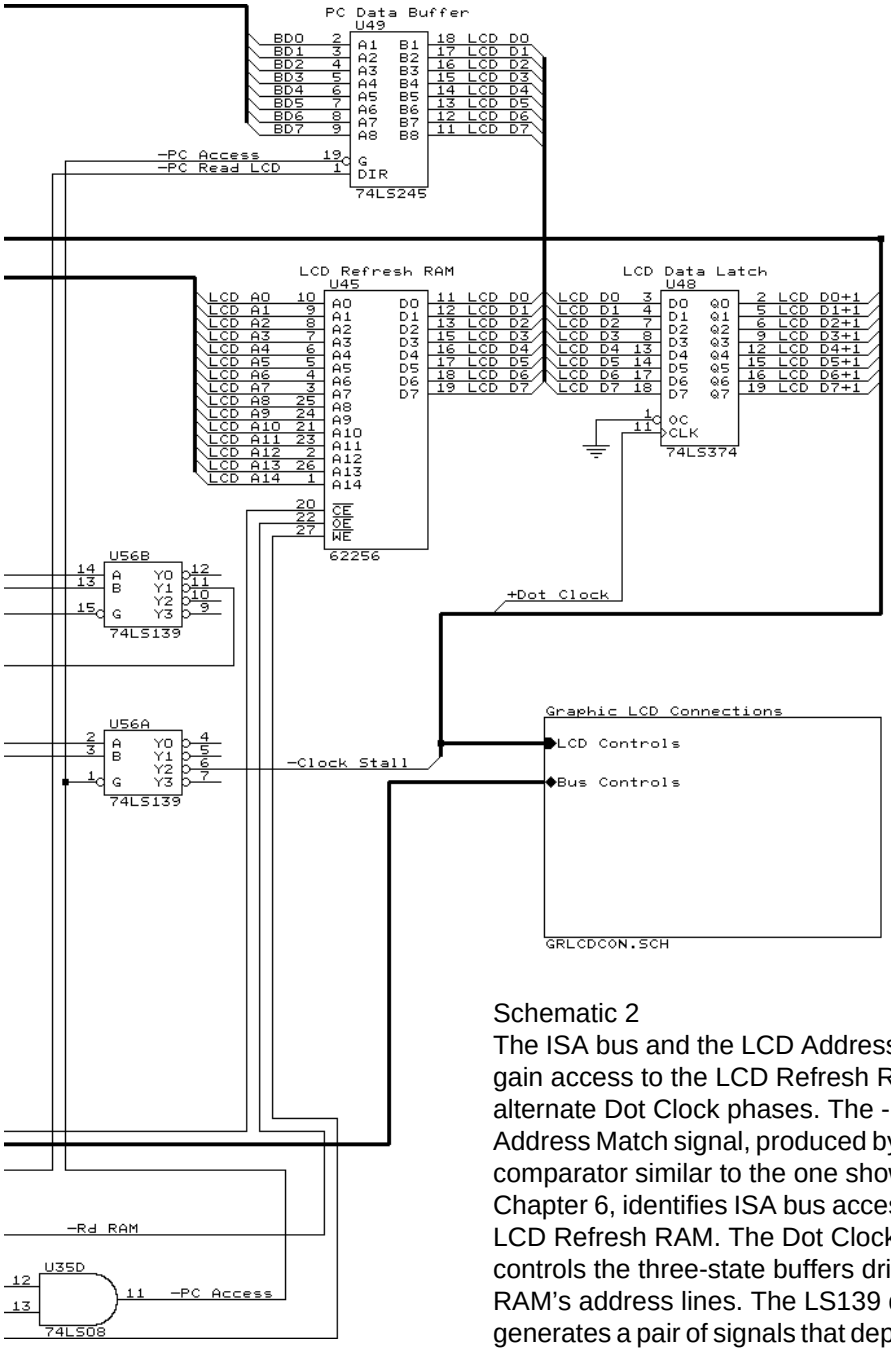
Note that the latch inside the LS590 is always one count *behind* the counter, because they share the same clock input. The latch captures its input value just before the counter changes on the rising edge of the clock input. Check the LS590 data sheet for more details, then ponder the timing diagram in Figure 1.

**Frame Sync** clears the counter address to zero at the beginning of each panel scan. The actual reset signal occurs during only a quarter of the **Dot Clock** period, thus ensuring that the LS590 counter will step from 0 to 1 on the next clock pulse. Figure 1 shows the timing: the reset pulse must be finished by the next **-Dot Clock** rising edge. Notice that the zero count is shorter than all the rest, due to the reset signal in the first quarter of the period.

The latch inside the LS590 solves a critical timing problem: the path length from the **Frame Sync** reset logic through the counters and RAM into the **LCD Data Latch** takes longer than 240 ns. Latching the address provides a stable signal for the entire LCD phase of **Dot Clock**, while delaying the RAM's output data by one cycle.

The RAM address alternates between the **LCD Address Counter** and the PC's ISA bus address. The LS590 latch outputs become active with **+Dot Clock** low, while U46 and U47, the **PC Address Buffers**, go active when **-Dot Clock** is low. Figure 1 shows the sequence of events; note how the LS590 latch ensures that the RAM address remains valid while **-Clear Counter** resets the **LCD Address Counter**.






Schematic 2

The ISA bus and the LCD Address Counters gain access to the LCD Refresh RAM on alternate Dot Clock phases. The -LCD Address Match signal, produced by a 74F521 comparator similar to the one shown in Chapter 6, identifies ISA bus accesses to the LCD Refresh RAM. The Dot Clock signal controls the three-state buffers driving the RAM's address lines. The LS139 decoder generates a pair of signals that depend on the Dot Clock phase and control other parts of the circuit.

The **LCD Data Latch**, U48, captures the **LCD Refresh RAM**'s output data when **+Dot Clock** goes high, 240 ns after the LCD's half cycle begins. This ensures stable data at the LCD panel when **+Dot Clock** goes low 240 ns later. The panels have various timing specs, with some requiring at least 150 ns of setup and hold time. Without this latch, the data at the LCD panel would disappear long before the panel captured it.

## Off by One...

Pop quiz!

1. According to the LCD panel specs, when does the **Frame Sync** pulse occur in relation to the first row of data?
2. In Figure 1, what is the RAM address of the value in the **LCD Data Latch** with **Frame Sync** active and what part of RAM appears on the top line?
3. What Zen koan does this situation suggest?

OK, keyboards down. The answers:

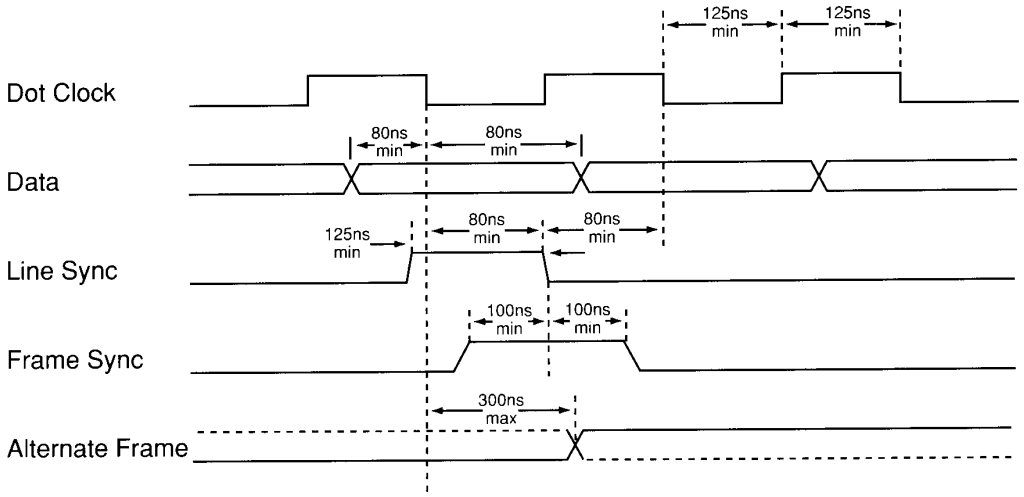
1. After 160 **Dot Clock**s, at the *end* of the first row of panel data.
2. 31999. The top line shows the bottom of the RAM data.
3. Ah, Grasshopper, consider the sound of one hand clapping… forehead. At least, that's the sound you'd hear if you hadn't drawn Figure 1 first.

Figure 2 presents the Optrex DMF651 640×200 panel pulse timing diagram, redrawn with my nomenclature. You will see three vital synchronizing relationships: **Line Sync** *vs.* **Dot Clock**, **Frame Sync** *vs.* **Line Sync**, and **Alternate Frame** *vs.* both **Dot Clock** and **Frame Sync**.

The falling edge of **Line Sync** must be more than 80 ns away from a falling **Dot Clock** edge. U37B delays the 82C54's **-Raw Line Sync** output until the next rising **Dot Clock**. Figure 1 shows that **+Line Sync +0.0** rises when the last data byte in each row appears in the **LCD Data Latch** and falls on the first byte of the next row.

**Frame Sync** must be high for at least 100 ns on either side of the first **Line Sync** falling edge in the frame. U37A synchronizes **-Raw Frame Sync** to the next rising +**Dot Clock** and U57 delays it by half a **Dot Clock** cycle. Figure 1 shows the result: the pulse called **+Frame Sync +0.5** lies neatly centered on the *falling* edge of **+Line Sync +0.0**, with about 240 ns setup and hold times. The other outputs of U57 delay **-Raw Frame Sync** even more, should your panel require a later signal. If not, replace U57 with a single flipflop.

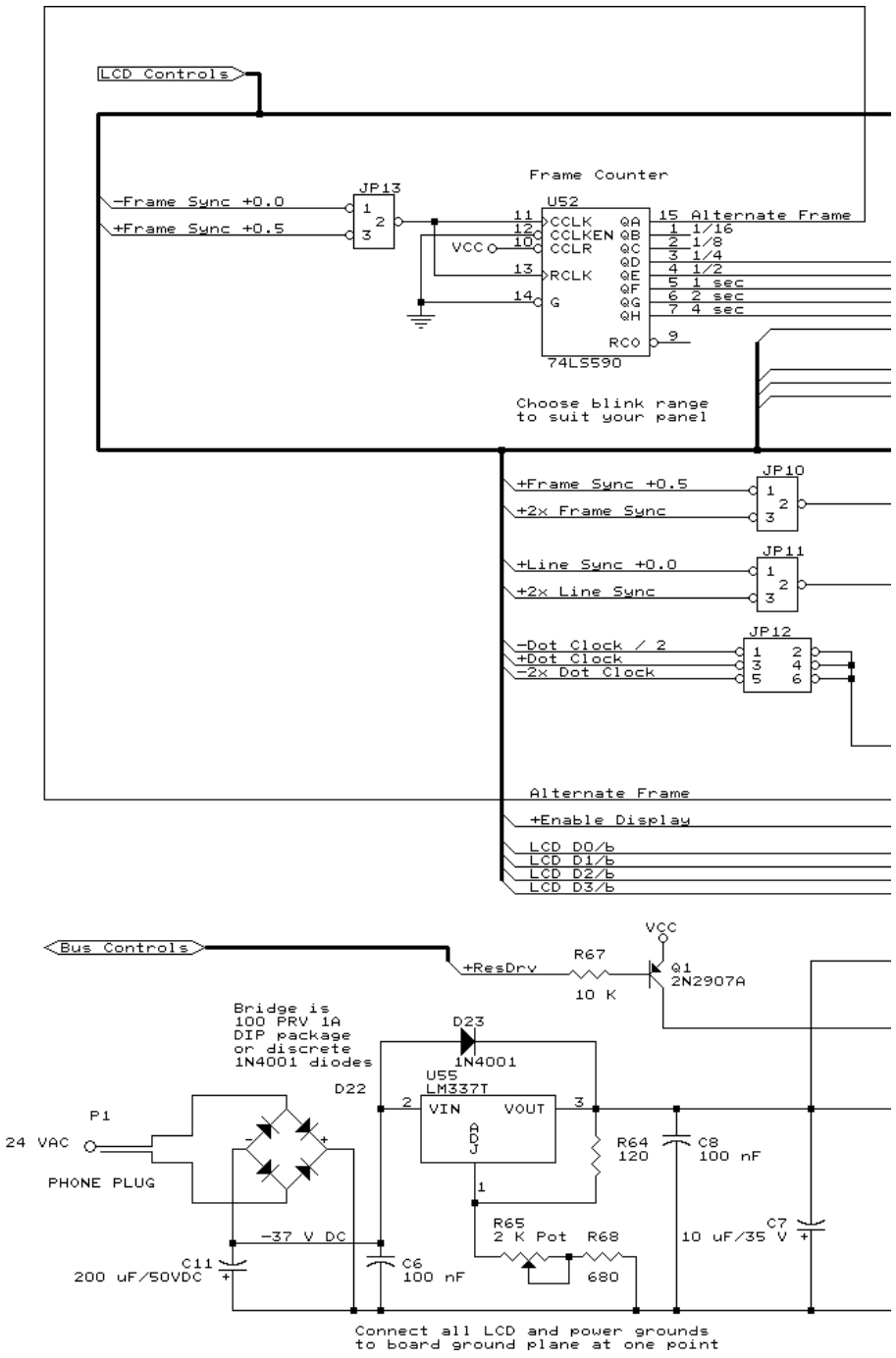
Figure 2
The Optrex DMF651 signal timing specifications dictate how the Graphic LCD Interface must operate. Notice that both data and Line Sync must be valid on the falling edge of each Dot Clock pulse, but Frame Sync's timing is related to the falling edge of Line Sync.

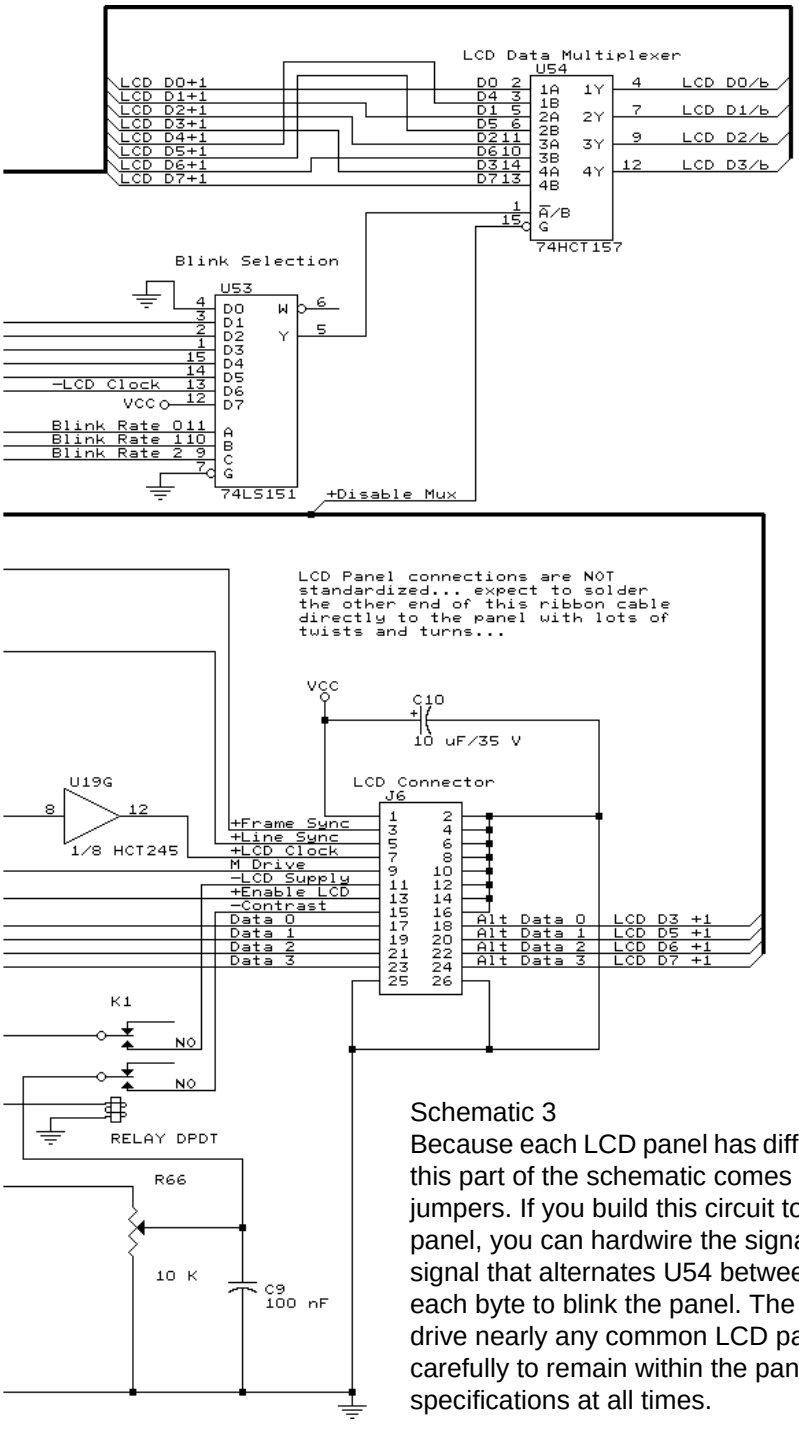



The **Alternate Frame** signal comes from U52 in Schematic 3. For the DMF651, this signal must change state within 300 ns of the **Dot Clock** falling edge at each **Frame Sync**, so I clocked U52 with **+Frame Sync +0.5**. Because the Hitachi LM215 refers **Alternate Frame** to the falling edge of **Line Sync** rather than **Dot Clock**, a jumper allows you to select **-Frame Sync + 0.0** instead.

U41 and U42 generate sync signals for panels that accept data every 240 ns. The double-speed **2x Dot Clock** changes the allowable width and alignment of the sync signals, leaving the basic Graphic LCD Interface hardware unchanged. The firmware sets up the **LCD Data Multiplexer** to switch between the two nybbles every half **Dot Clock** cycle and provide the data bits on time. I'll go into more detail about the blinking functions later.

As I mentioned earlier, you must drive all the LCD signals with CMOS logic gates to meet the panel's minimum $\mathbf{V_{IH}}$ spec. Because I used several spare gates from other sections of the FDB's logic, some of the LS gates mentioned earlier in this book metamorphosed into HCT. In actual practice, LS gates will probably work, if only by the grace of wide tolerances… which is bad practice.






Schematic 3

Because each LCD panel has different requirements, this part of the schematic comes festooned with jumpers. If you build this circuit to drive a particular panel, you can hardwire the signals. U53 selects a signal that alternates U54 between the nybbles of each byte to blink the panel. The power supply can drive nearly any common LCD panel. Adjust it carefully to remain within the panel's voltage specifications at all times.

## Porting a RAM

When **+Dot Clock** is high, the PC controls the **LCD Refresh RAM**. The outputs of U46 and 47, the **PC Address Buffers**, become enabled during that half cycle. U49, another LS245, shuttles PC data to or from the RAM. These buffers run in series with those already on the FDB.

Access from the PC side of the interface encounters a gotcha: the LCD's **Dot Clock** cycle is not synchronized with ISA bus read and write cycles. If **+Dot Clock** goes low near the end of a bus cycle, the CPU will read invalid data or write junk into the RAM as the PC's buffers turn off prematurely and the LCD circuitry takes control. Obviously, this is a Bad Thing.

Solving this problem requires a delay: either the PC must wait until **+Dot Clock** goes high or the LCD panel must wait until the ISA bus cycle finishes. I chose the latter course, because you can pause an LCD clock for a short time without visible effect. You can do it the other way if you prefer, bearing in mind that the ISA bus runs slowly enough already and more delay might affect your program's ability to update the entire buffer in a timely manner. Decide which is more important in your application, then measure the results to be sure it does what you want.

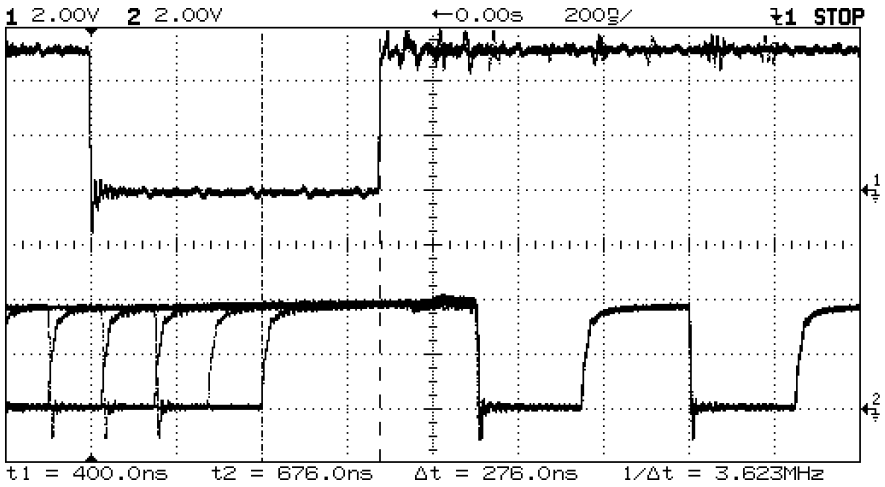
The multiple exposure traces in Photo 1 show how this works. The top trace, **-SMemW**, controls the ISA write cycle timing. The lower trace, **+Dot Clock**, normally appears as a 480 ns square wave. Notice that **+Dot Clock** is always high when **-SMemW** rises, even though it may not be synchronized when the cycle begins. Normal **Dot Clock** cycles resume after **-SMemW** goes high at the end of the bus cycle.

A complete 8-bit ISA bus memory write access requires six **SysClk** cycles, with **-SMemW** active for about the last 4.5 cycles. Because **Dot Clock** occupies four **SysClk** cycles, the maximum delay until **+Dot Clock** goes high after **-SMemW** becomes active will be three **SysClk** periods.

U56A monitors the double-speed **2x Dot Clock**, **Dot Clock**, and the (negative active) OR of **-SMemW** and **-SMemR**. When **+Dot Clock** goes high during a PC access, U56A's output freezes the **SysClk** divider flipflops and holds them unchanged until the PC access finishes.

The shortest RAM access thus occupies about 1.5 **SysClk**s. Allowing for buffer and logic delays, a 120 ns RAM is just fast enough. I suspect some systems may require a 100 ns RAM or 74F buffers instead of 74LS parts. In any event, work out your system's timings and see how close it comes.

Photo 1
An ISA bus read or write access stalls +Dot Clock in the high state until the operation completes. The top trace, -SMemW, stays active for about 4.5 SysClk cycles. Dot Clock, shown in the bottom trace, is not synchronized with -SMemW when the write cycle begins, so it may end at any of four times after the falling edge of -SMemW. The RAM access starts when +Dot Clock goes high and ends when -SMemW goes high, allowing enough time for a complete LCD Refresh RAM access, even from the latest Dot Clock.




At worst, you may need an additional wait state or two on each PC access. Don't make the mistake of stalling both **`Dot Clock`** *and* the ISA bus, though… if *both* sides of the interface stop, *neither* will restart!

Even though you can stretch a **`Dot Clock`** cycle by up to three **`SysClk`**s when it collides with a PC access, you'll never see it as long as it's rare enough. One of my test routines writes data into the RAM while toggling a parallel port bit that provides scope sync: one byte every 33 µs affects an invisible 1.5% of all **`Dot Clock`** cycles.

On the other hand, another test routine runs a `REP MOVSB` that reads all 32 KB at 960 ns/byte. That *severely* distorts the **`Dot Clock`** cycle, holding it high for 720 ns and allowing one 240 ns low half cycle between each access. The refresh rate drops to 33 Hz and the panel develops an obvious flicker.

But remember, even if you read or wrote the entire 32 KB with a `REP MOVSB`, you wouldn't do it in a loop, as I did in the test routine. The LCD panel's optical

response falls in the 100-200 ms range, so it doesn't have time to respond to 30 ms of furious writes. Those new dots take another hundred milliseconds or so to replace the old ones. You'll see both types of access in Chapter 15, where we will generate dots in the buffer one by one, then copy the entire buffer into system RAM with a block move.

Do note, however, that this interface is *not* suitable for DMA or busmastering ISA bus board access, because the timings depend on a specific relation between **`SysClk`** and **`-SMemW`** that will be valid only during system board accesses. If your project demands a more versatile interface, check Solari's books for the grim details.

Even when all your logic works correctly, you won't see anything on the LCD panel unless you apply the proper LCD drive voltages. I won't mention how often I've suffered palpitations at the sign of a blank panel, only to realize I left the power supply lead dangling once again.

## Practical Power

The power supply circuitry shown in Schematic 3 certainly counts as overkill. Because any particular panel draws perhaps 20 or 30 mA at a single voltage, there's obviously no need for a wall wart transformer, a huge capacitor, and a 1.5 A, 15 W voltage regulator in a TO-220 case. Suffice it to say you can find easier and more compact ways to generate drive voltages for any specific LCD panel. Check the Maxim data books and Web page for hints and tips about their line of single-chip LCD bias supplies.

But, when you're working with a variety of panels, it makes perfect sense to build a versatile supply using parts from your junk box. Even better, you can probably find a power supply designed for precisely this purpose from any of the surplus parts dealers listed in the Sources appendix… at an attractive price, too.

Relay K1 marks my concession to the variety of power supply sequencing specifications found in the data sheets. All of the panels I've seen remain happy if you apply +5 V first and remove it last, which the DPDT relay accomplishes by disconnecting the drive voltages. The ISA bus **`+ResDrv`** signal goes active when the power supplies fall out of tolerance or you reset the system. That ensures the +5 V supply always becomes active first and remains active last.

**CAUTION!** Set the LCD drive voltage *before* you connect the panel, then monitor it with a voltmeter as you adjust the trimpot. The supply shown in Schematic 3 can apply enough voltage and current to destroy a panel with the twist of a screwdriver. Be overly hypercautious here, if nowhere else.

## Greater & Lesser Arrays

The circuitry I've just described can handle panels with 480 ns **Dot Clock**s. The situation will be slightly different for panels requiring 240 ns or 960 ns clocks, but the hardware requires few changes. You can build only the sections for your panel and omit the jumpers festooning my prototype board.

The Sharp LM64015T 640×400 panel I mentioned in the previous chapter expects 320 data transfers on each of 200 rows, synchronized with a 240 ns clock signal. The **LCD Refresh RAM** still delivers eight bits every 480 ns, but the firmware sets up the **LCD Data Multiplexer** to switch between the two nybbles on each half of the **Dot Clock** cycle. As a result, the panel's dots line up quite nicely with the data bytes: the high and low nybbles come together at last.

The **Frame Sync** and **Line Sync** pulses last 240 ns and are synchronized by the double-speed **2x Dot Clock**. I used U41, an LS139 decoder, to control the two flipflops in U42 with the result shown in Figure 1, and drove the panel with **-2x Dot Clock** to place a falling edge in the middle of each nybble.

A Sharp LM641481 640×480 panel I found resembles the LM64015T, except that the extra 40 (not 80… remember how the panel hardware works?) rows reduce the refresh rate to about 54 Hz. Unfortunately, this high-performance panel has a brilliant cold cathode fluorescent tube backlight and flickers badly at 54 Hz. As with the LM64015T, the CCFT inverter seems particularly hard to find from the usual surplus sources.

Also, remember that a 640×480 panel requires 38400 bytes of RAM, which exceeds the standard Graphic LCD Interface design. Although it's not shown on the schematic, you can piggyback two 32 KB RAMs with their respective **-CE** inputs driven by **+LCD A15** from U44 and **-LCD A15** from an inverter. Disconnect ISA bus line **A15** from the F521 comparator to grant the PC access to all 64 KB between D000:0000 and D000:FFFF.

In contrast to that vast array of dots, the Hitachi LM215 480×128 panel is, electrically, a quartet of dinky 240×64 (!) arrays. Four data bits arriving every 960 ns refresh it at about 68 Hz. Each data bit drives a separate quadrant, with 240 half-speed **Dot Clock/2** pulses on each of 64 rows. Use the **-Dot Clock/2** jumper and run the **Alternate Frame** divider from **+Frame Sync + 0.5**.

The LM215 displays only 15360 nybbles of data, so you can use the high order nybbles in the **LCD Refresh RAM** for blinking as usual. Remember that, because the panel runs at half speed, you must duplicate the dots in successive even-and

odd-numbered bytes for the first 30720 RAM locations. The test pattern for this panel looks somewhat different than the others in order to cover all four quadrants.

You will certainly find a different assortment of panels today than I did when I set out to design this project. The most significant change comes from the PC laptop market, where 640×480 panels are now pretty much standard equipment. As a result, the prices of those panels have dropped rapidly and they're no longer nearly as exotic as they once were. Make sure that you get the specs for any panel you purchase, particularly if you go the surplus route, because you cannot figure out many of the essential connections without the documentation.

## Release Notes

***Reminder!*** Be careful with that LCD power supply. Crosscheck the LCD pin numbering with an ohmmeter: the supply voltages and grounds should appear on the panel's filter caps with the appropriate polarities, right? As woodworkers say, think thrice, measure twice, and cut once… smoking a new panel can really mess up the rest of your day.