12 Lots'a Dots

You probably have a tempting embedded PC application for those big LCD panels you see in the ads, but the phrase "external controller required" translated into "Well, maybe, one of these days…" Figuring out how to use a bitmapped graphic LCD made the project hard enough that you never got around to it, right?

I'd been thinking about them for a while, too, for a series of firmware projects that can take advantage of a big display that *isn't* a CRT, *doesn't* require a serial port, and *must* function with a VGA board also in the PC. As you'll see in the next few chapters, a handful of logic gates, plus a dollop of firmware, can convert an LCD panel into a bitmapped graphic and alphanumeric display. Best of all, you won't need a specialized LSI widget between you and the dots.

Overall, the Graphic LCD Interface is about as complex as the rest of the Firmware Development Board taken together. It summarizes most of the techniques we've used so far and illustrates how you can combine firmware and hardware to implement a complete design.

I'll start by describing how graphic LCD panels work, present some condensed specs for several typical panels, and go over the interface's hardware block diagram. You'll be in a better position to understand why the Graphic LCD Interface works the way it does when you see what all the signals do after they get to the panel. If only our real-world projects had such bounded properties, we'd have fewer failures!

In the next two chapters, you'll see the interface hardware schematics and the test code that checks it out on your system. Because of the variety of panels floating around out there, the circuitry sports an unusually large number of options and jumpers. I'll also describe some embellishments you may want for a few oddball panels that lie just outside the standard hardware's capabilities.

With the hardware up and running, we will explore some bitmapped graphics, using Conway's Game of Life to generate the dots. Each type of panel has a different memory layout, making the task more complex than it seems and giving us a good reason to use modular firmware. Fortunately, graphic output devices are easy to debug when your errors appear right in front of your eyes.

Finally, I'll show how to generate text characters on a bitmapped panel, a trick that should come in handy for status displays on your embedded projects. The LCD firmware uses standard ANSI cursor control codes, allowing the same output data stream that reports status through the serial port to a PC comm program on your

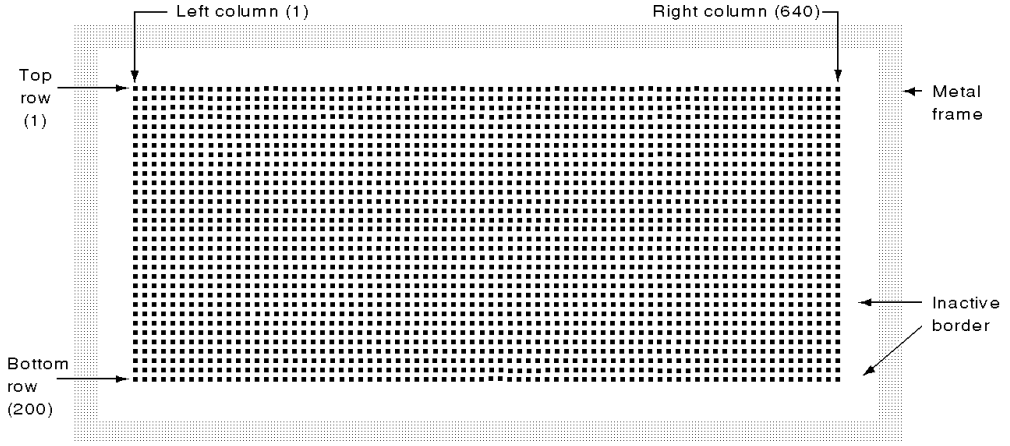
host system to drive the LCD panel. If your target system has a 640×400 panel, you'll see 50 lines with 80 characters each. That should be enough, right?

Dots in Ranks and Rows

Graphic LCD panels are fairly easy to understand, but the nomenclature certainly doesn't provide any assistance. Each manufacturer uses different names for the control signals and I've even seen one signal with two names in a single data sheet. I'll use signal names that I find descriptive, but you should plan on a little data sheet spelunking to match them up with your panel.

A graphic LCD panel consists of a rectangular array of dots similar to Figure 1. Generally, the grid has its larger number of dots running horizontally, so we'll call an array with 640 dots along each of 200 rows a "640×200" panel. It's reasonably easy to produce a 200×640 portrait display, should you need one for your next project, using firmware to rotate the character bitmaps a quarter-turn. As the saying used to go, "We control the vertical".

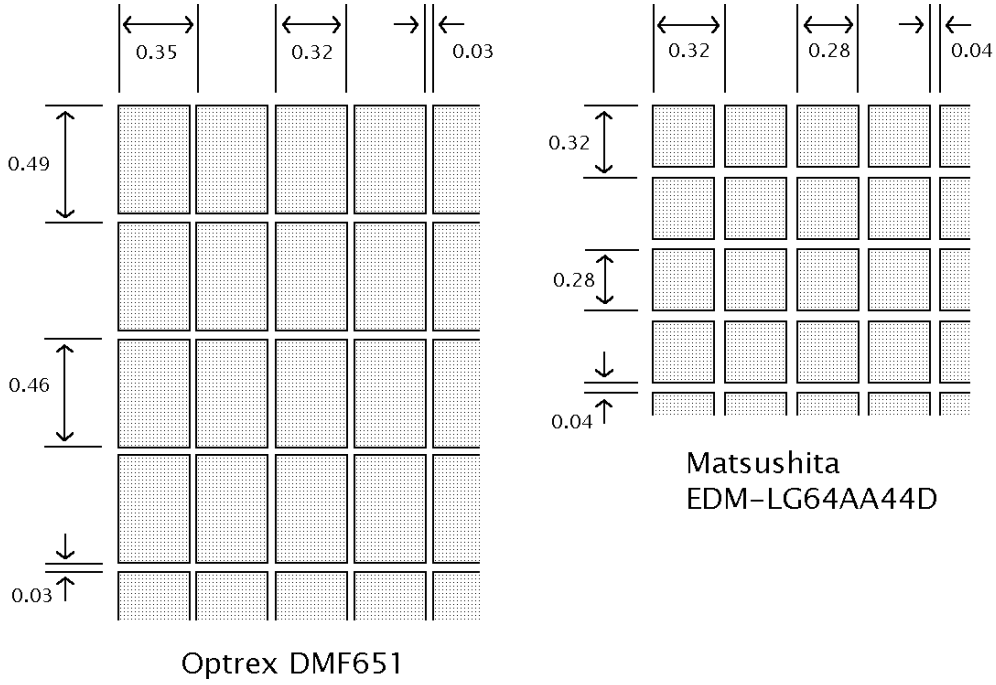
Figure 1
A graphic LCD panel is essentially a thin double-pane window: you actually see the reflector or backlight atop the circuit board underneath the panel. Transparent electrodes laid out in horizontal rows on one pane and vertical columns on the other delineate the dots at their intersections. The liquid crystal syrup between the panes reacts to the applied electrical field at each dot by rotating the transmitted light's polarization; fixed polarizers outside the panes cause the dots to appear dark or light as the dots change their polarization. The metal frame clamps the whole affair to the underlying circuit board and ensures good connections to the hundreds or thousands of column and row drivers.



Although older panels had rectangular dots, most recent panels present square dots in equally spaced rows and columns. The small gap between each pair of dots is inactive, so, unlike pixels on a CRT, each dot has crisp, distinct edges. Figure 2 shows the dot dimensions for Optrex DMF651 640×200 and Matsushita EDM LG64AA44D 640×400 panels.

The panels we'll use for this project have a binary interface: each dot must be either completely ON or OFF. The newer VGA and SVGA panels can present continuous grayscale or color images, but driving those displays is a whole 'nother subject. For now, each dot on our panels represents a single bit: a DMF651 displays 128,000 bits. Yes, that's a nice round decimal number with lots of zeros, neither the binary 128 kilobits nor its decimal equivalent of 131,072 bits.

Figure 2
The width of the transparent row and column electrodes determines the shape of the dots at their intersections. Because the electrodes must be isolated from each other, every pixel has an inactive border surrounding it. Recent panels, typically those with 400 or more rows, have square dots. This simplifies life for graphic programmers: displaying *round* circles and *square* squares take no special effort.



A binary 1 bit may make its corresponding dot either transparent or opaque, depending on the liquid crystal chemistry, the driving electronics, and the arrangement of the external polarizers and reflectors. Because we have complete control over the data, dark dots on a light background or vice versa require no circuit changes. However, applying a `NOT` operation to every bit cannot change the inactive border around each dot. In general, you should set up the data so that a 0 bit produces a dot that matches its border and a 1 bit is distinctly different.

Contrary to popular opinion and despite what your eyes tell you, those bits are not all active at the same time. You cannot just write 128,000 bits into the panel and go about your business: the panel itself does *not* have a frame memory. You must send the same bits to the panel at least 60 times per second to produce a stable, flicker-free image. Surprised?

I recall a discussion with someone who tried driving a big, graphic LCD directly from an 8031 microcontroller's output bits. Once we went over the code's timing, he realized what was wrong: his hardware refreshed the display at about 3 Hz! An 8031 just can't supply four bits every 500 ns, making some additional support hardware a distinct necessity. Whether a big, bitmapped panel made sense in an 8031 application was another question that we didn't explore at the time.

Anyhow, an "external controller" is just that "additional support hardware" squashed on a single chip. It manages all the control signals, refreshes the panel data, and provides a convenient CPU interface. You'll find two catches: any given controller can handle only a subset of the panels out there and they all come in those minuscule, awkward, surface-mount packages. What fun is that?

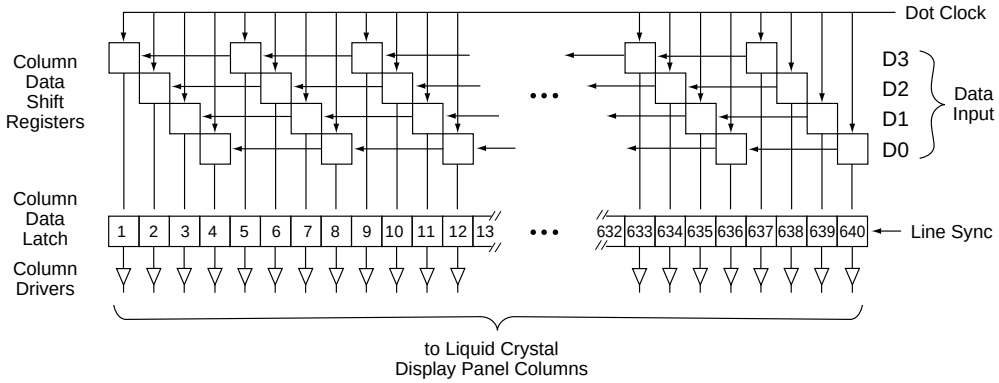
After you decide on the refresh rate, simple arithmetic provides the panel data rate:

(128,000 dots/frame) × (60 frames/second) = 7.6×10^6 dots/sec = 1 dot per 130 ns

Unlike a CRT that lights up only one pixel at a time, an LCD panel accepts several bits in parallel. For example, the DMF651 clocks in four bits (that's one *nybble*, pronounced with the *y* from *nymph*, not the *y* from *nylon*) every 520 ns or so. Transferring the 640 bits appearing in each row requires only 160 clock cycles and displaying the entire 200 line frame takes exactly 32,000 **`Dot Clocks`**.

Rather than presenting each nybble as it arrives, the DMF651 accumulates them in a 4-bit-wide, 160-element-long shift register. When that register contains all the bits for the next row, a single **`Line Sync`** pulse transfers them to a 640-bit Column Data Latch leading to the panel's column drivers. Figure 3 shows the connections for the beginning and end of this circuit.

Figure 3
A shift register accumulates 160 groups of four bits to drive all 640 dots in a single row simultaneously. The Dot Clock sets the basic timing for the entire panel, because all the other signals are defined in terms of its period and transitions. The Line Sync pulse transfers data from the shift register to a 640-bit parallel latch that holds the output stable while the next 640 bits arrive.



Yes, I know it's sadly inconsistent to talk about LCD *rows* and ***`Line` Sync*** signals. Unfortunately, my notes, sketches, and code started out that way and, despite my best efforts, it's much too late to change now. Take this as an object lesson about both the importance and difficulty of planning ahead. Check those LCD panel specs for more of the same, too.

Because the panel displays the dots in each row until all of the bits for the next row arrive, each row (and, thus, each dot) has a duty cycle of 1/200 rather than 1/32000. As you might expect, making a dot appear ON when it's active only 0.5% of the time presents a considerable challenge; don't even think about a 0.003% duty cycle. Hats off to the engineers who make LCD panels work at all, let alone as well as we've come to expect.

LCD panels don't implement row selection with a binary counter and decoder the way you might expect, either. The panels include another shift register, illustrated in Figure 4, behind the row drivers. Successive **`Line Sync`** pulses clock this register and pass the **`Frame Sync`** pulse through each flipflop in the chain. Unlike the 640-bit Column Data Latch shift register, the Row Select register has no output latch, because the **`Frame Sync`** pulse activates only one row driver at a time. We will meet some exceptions to that rule later in this chapter.

A **`Frame Sync`** pulse accompanying a **`Line Sync`** pulse marks the first display line. Because **`Line Sync`** pulses occur at the end of each line, the **`Frame Sync`** pulse occurs *after* the first row of data arrives. Figure 5 shows the relationship between the various pulses. It should go without saying that not all panels operate quite the same way, but you can see the general ideas.

Figure 4
The outputs of a 200-bit shift register drive the LCD panel rows. The Frame Sync pulse passes from one latch to the next on each Line Sync pulse, activating only one row at a time. To achieve a steady display, you must refresh each row often enough to prevent visible flicker, which implies a rate that typically exceeds 60 Hz. Each row occupies 1/200 of the total frame time, or about 80 µs in each 16 ms. The jolt applied to the liquid crystal material turns the dot ON within that 80 µs and it gradually fades OFF during the 15,920 µs until the next pulse.

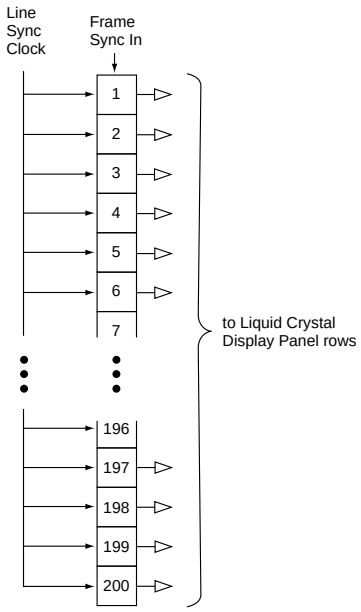
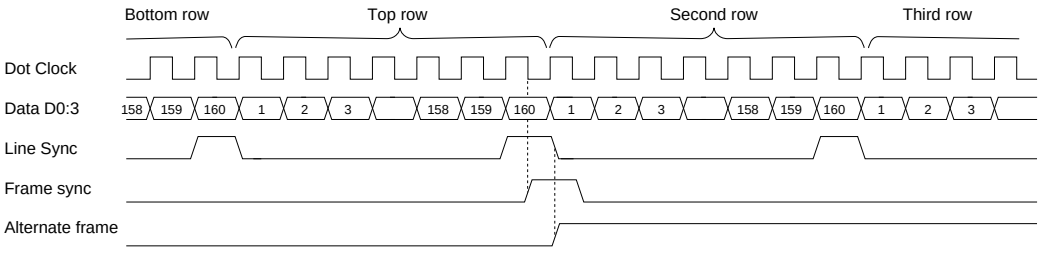


Figure 5
Each falling edge of the Dot Clock signal transfers data into the column data shift registers. If the Line Sync signal is high when Dot Clock goes low, the shift register contents transfer to the column driver latches. The Line Sync signal also clocks the Frame Sync pulse through the row driver shift register shown in Figure 4. Note that Frame Sync is active at the *end* of the top row, not between the top and bottom rows as you might expect from your experience with CRTs.

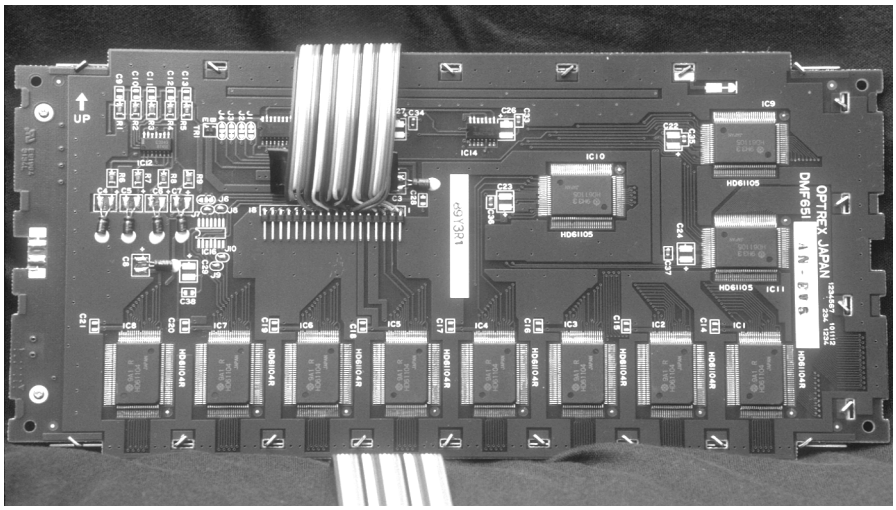


The DMF651 requires an additional signal that alternates from frame to frame. To produce this signal, which I call **`Alternate Frame`**, you just toggle a flipflop on each **`Frame Sync`** pulse. More recent panels generate the signal internally and I suspect this has more to do with the size of the driver IC packages than anything else. After all, it's easy to add an LSI flipflop when you have a spare output pin, but impossible to justify adding another whole chip for just one bit.

Using shift registers for both the row and column logic circuits allowed the LCD engineers to split the circuitry into identical units, precisely the right tactic for good LSI chip design. For example, the DMF651 shown in Photo 1 has eight Hitachi H61104 column driver chips and three H61105 row drivers. Each H61104 contains 20 elements of the four-bit column shift register and drives 80 columns. Each H61105 has 80 row selection bits, leaving half of the last chip unused.

The actual process of converting row and column selection bits into visible dots lies, mercifully, hidden in those LSI driver chips. Several different schemes, each applying to a different panel configuration, all boil down to applying a high voltage to the dots at the intersection of each active column with the currently selected row, while not applying quite so much *oomph* to all the other dots. Under the influence of that jolt, the liquid crystal compound twists the transmitted light's polarization and the dot becomes either opaque or transparent, depending on the panel's external polarizer and reflector arrangement.

Photo 1
This Optrex DMF651 640x200 LCD panel uses eight Hitachi H61104 column driver chips and three H61105 row drivers.



Because of the high voltage required to drive the chemistry, graphic LCD panels are *not* friendly 5 V devices. In addition to the usual +5 V logic supply and ground, they require an LCD drive voltage between about -9 and -30 V at perhaps 25 mA. Some displays use both a fixed, relatively high current, LCD bias voltage and a separate contrast adjustment voltage that draws much less current.

You must adjust the negative voltage to compensate for the inevitable temperature changes that affect the liquid crystal material's optical response. A trimpot works well, if your application can stand a manual adjustment, or you can use a DC to DC converter to produce an adjustable negative voltage directly from the +5 V logic supply. In any event, recheck the negative bias voltage when you can't see any dots. Trust me, you'll wonder why the panel is completely blank, too.

Most of the LCD panels you see here originally appeared in laptop computers or similar low power widgets. Their CMOS logic reduces power consumption, but CMOS logic levels make no concession to TTL drivers. For example, the V_{IH} for most panels is typically 0.8 V_{CC} (where V_{CC} is the panel's supply voltage), which translates into 4.0 V. One panel expects V_{IH} levels exceeding 0.9 V_{CC}! You should drive the interface signals with HCT or HCTLS gates, because ordinary LSTTL has a minimum V_{OH} spec of 2.4 V, well below the CMOS threshold.

BEWARE! The panel's logic-level inputs and outputs have *no* protection against those negative LCD bias voltages. Despite the fact that the negative supply lines may be sandwiched between logic signals on the LCD connector, you must *never* short those adjacent pins, not even once, not even when your scope probe slips. Word of advice: buy two LCD panels and save on shipping…

By the way, the Firmware Development Board drivers have no protection, either. One slipped probe can blow away both the panel and your own logic. That's true of nearly all panels and their control logic, as this hardware is really meant for OEM production, not the sort of tinkering and experimentation we do here.

Panels specs also present different sequences in which you must apply and remove their supply voltages and logic signals during startup and shutdown. The penalty for disobedience can be your panel's death due to SCR latchup as its CMOS logic incinerates itself. If you're designing a specific panel into your project, you can meet its peculiar demands, but I don't know of a good general solution that can handle a wide variety of panels. The FDB circuit includes a DPDT relay that disconnects the LCD drive voltages when the ISA bus **ResDrv** signal goes active, but that surely doesn't meet all the specs. So far, though, so good.

Finally, not only do the manufacturers use different signal names, bias voltages, and power sequencing, but each LCD panel also sports a unique connector. Sometimes,

Figure 6
Each of these graphic LCD panels sports unique electrical and connector specs. This table summarizes the important characteristics and pin functions for some of the panels in my stash. An "n/c" entry means a pin is not used, while a blank entry means the connector doesn't have that pin. A separate cable carries power for the backlight.

Panel Spec	Optrex DMF651	Matsushita LG64AA44D	Sharp LM64015T	Toshiba TLY-365-121	Hitachi LM215XB	Epson EG7004
Dots	640 x 200	640 x 400	640 x 400	640 x 200	480 x 128	640 x 400
Bits	4	8	4	4	4	4
Col Drivers	640	640	640 x 2	640 x 2	240 x 2	640 x 2
Row Drivers	200	200 x 2	200	100 x 2	64 x 2	100 x 2
Clocks/Row	160	160	320	320	240	160
Dot Clock Pd	480 ns	480 ns	240 ns	480 ns	960 ns	240 ns

Connector Pinout

Pin	Optrex DMF651	Matsushita LG64AA44D	Sharp LM64015T	Toshiba TLY-365-121	Hitachi LM215XB	Epson EG7004
1	Bezel Gnd	+5 V	Frame Sync	Bezel Gnd	D1 UL	+5 V
2	Line Sync	Bezel Ground	Line Sync	n/c	D2 LL	Ground
3	Dot Clock	Dot Clock	Dot Clock	Frame Sync	Frame Sync	-Contrast
4	Alt Frame	Enable	n/c	Line Sync	Alt Frame	Line Sync
5	-Contrast	Frame Sync	n/c	Dot Clock	Line Sync	Alt Frame
6	+5 V	Line Sync	+5 V	Ground	Dot Clock	Enable
7	Ground	Ground	Ground	D0	D3 UR	Row Shift
8	-23 V	n/c	Adj -21 V	D1	D4 LR	Frame Sync
9	D0	D0 upper	D0	D2	+5 V	Dot Clock
10	D1	D1	D1	D3	Ground	Col Enable
11	D2	D2	D2	Ground	-10 V	D0
12	D3	D3	D3	+5 V	-Contrast	D1
13		-22 V		-Contrast (?)		D2
14		-Contrast		Adj -22.5 (?)		D3
15		Ground		Ground		
16		D0 lower				
17		D1				
18		D2				
19		D3				
20		Ground				

different panels from the same manufacturer have different connectors. Figure 6 summarizes what I know about the panels I've tested, along with my pin name translations. The Graphic LCD Interface uses a 2×13 ribbon cable header that doesn't match *any* of the panels, but I got pretty good at soldering wires directly to panel connectors. Hint: don't waste your time trying to find the mating connector for your panel, unless, of course, you're going into production.

Seeing the Big Picture

The circuitry we'll explore in the next chapter serves as, perhaps, an example of retro design: the *right* way to do it requires the exact LSI controller for the particular panel you're using. But, because this book shows you *how* things work, I don't feel too badly about illustrating key points with a handful of TTL chips and a lot of firmware.

As with the ISA bus interface logic we built in Chapter 3, you can certainly reduce the number of discrete chips using programmable logic: an FPGA seems like a nice fit for the logic in Figure 7. Once again, I used TTL gates to keep things simple for folks who don't have access to that kind of hardware. If you've got the capability, see just how small you can make all this logic.

With that in mind, Figure 7 shows the overall Graphic LCD Interface block diagram. The PC sees the interface as a 32 KB block of RAM, called the **`LCD Refresh RAM`**, and a write-only output port. The LCD panel sees the interface as data bits and control signals. I'll start on the LCD side, because those functions determine how the PC side must operate.

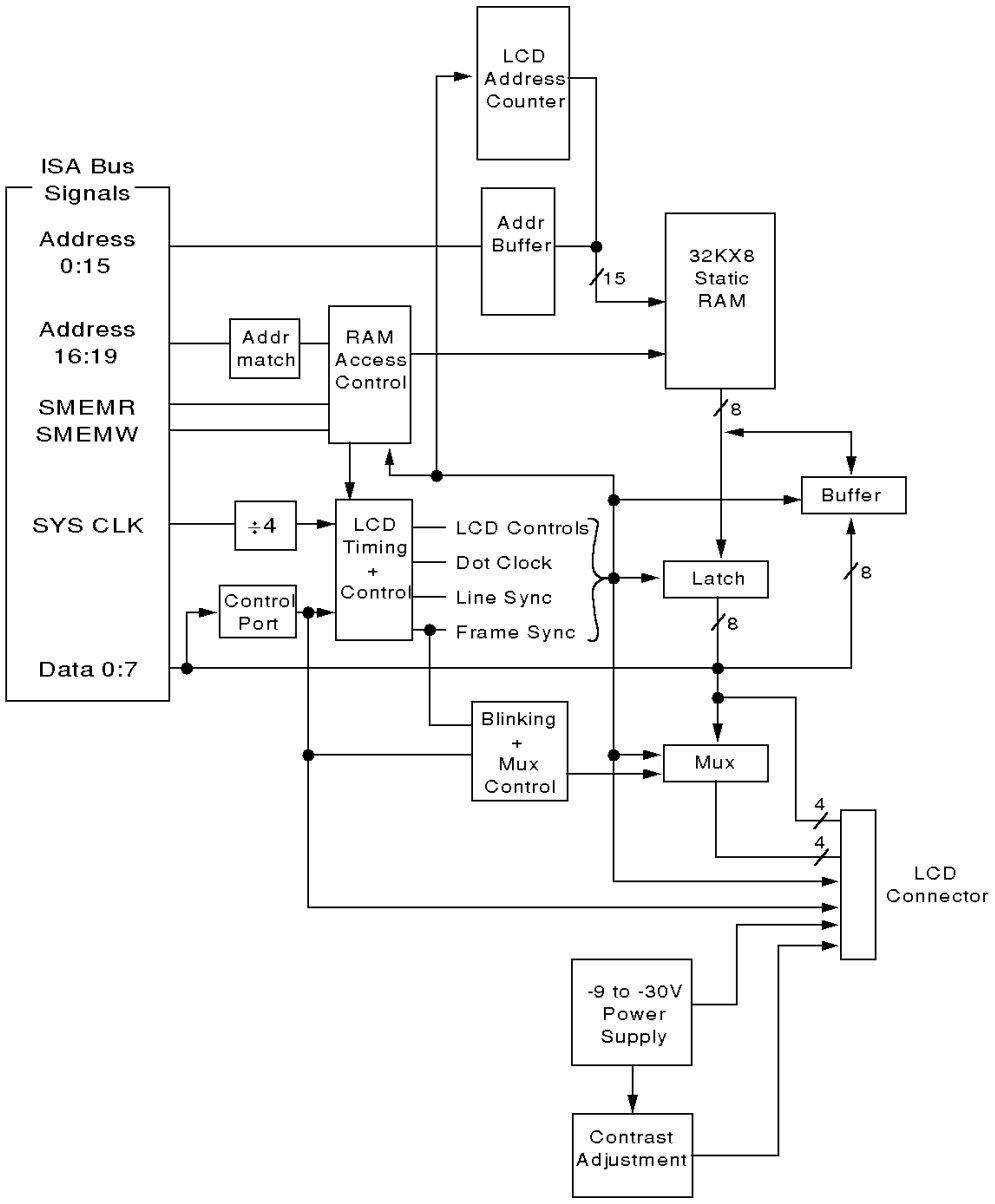
Displaying one 640×200 frame requires 32,000 clock cycles. Even though each cycle transfers only four bits, a byte-wide, 32 KB (32,768 decimal) static RAM chip is an obvious choice. If we store the dots in the low-order four bits of each byte, a simple 15-bit address counter will extract them in the right order.

The LCD **`Dot Clock`** period must be about 520 ns to refresh the entire display at a 60 Hz rate. A key part of this design fell into place when I realized a 480 ns **`Dot Clock`** would provide 65 Hz refresh. You should notice that 480 ns comes from an ISA Bus magic number: divide the 120 ns **`SysClk`** signal by four, thus increasing the period by a factor of four, and there you have it.

Most LCD panels specify a refresh rate between 60 Hz and 80 Hz. For a 200-line panel at 65 Hz, each line would take

(200 lines/frame) × (65 frames/sec) = 1 line every 77 µs

Figure 7
The Firmware Development Board's Graphic LCD Interface forms a dual-ported RAM. The PC can read and write data while the LCD panel displays dots fetched from the same location. The two sides of the interface access the RAM on alternate Dot Clock phases, so the operations aren't really simultaneous. The output multiplexer and blink logic include additional circuitry (and jumpers!) to support a variety of LCD panels.



and, with 160 **Dot Clock**s per line, the clock period works out to about 480 ns. See how to apply that to your panel?

The panel specs also list a minimum **Dot Clock** period, typically in the 160 ns to 330 ns range, or the corresponding maximum clock frequency. Running the clock at 480 ns stays comfortably above the minimum period, but there's another factor we must consider.

If each **LCD Refresh RAM** access requires the entire 480 ns available in each **Dot Clock** cycle, the PC has no time to write or read data. We must somehow implement the **LCD Refresh RAM** as a dual-ported memory, because forcing the firmware to check status flags simply won't provide the bandwidth required for a bitmapped display. You may recall the gyrations required to prevent visible snow on the old CGA graphics adapter… we don't want to repeat *that* mistake.

The Graphic LCD Interface simulates simultaneous PC and LCD access by using alternate 240 ns phases of each 480 ns clock cycle. While **Dot Clock** is high, the PC can access the RAM through the ISA bus address and data buffers as usual. When it's low, the LCD has full access for its address counters and data latch.

The LCD panel must receive stable data throughout the entire 480 ns cycle, however, so **Dot Clock** captures the RAM output data in a latch. Most panels accept data on the falling edge of **Dot Clock**, giving the bits nearly 240 ns of both setup and hold time. The latch holds the data stable while the **LCD Refresh RAM** processes PC accesses from the ISA bus during the next 240 ns phase, when **Dot Clock** is high.

Although the DMF651 uses only four data bits, it seemed a shame to waste half the RAM. A multiplexer after the latch selects either the high or low nybble under control of a signal from the Blinking and MUX Control logic. That signal, picked from one of the outputs of an 8-bit counter driven by **Frame Sync**, switches the multiplexer between its two 4-bit inputs every 1/16 to four seconds.

A little firmware can easily implement nearly any blinking scheme you'd like, because the two RAM nybbles can hold entirely independent data. If you fill the high nybble with zeros, the ON dots in the low nybble blink. Fill it with ones to get a blinking background while the data dots remain ON. Duplicate the low nybble in the high nybble, complement the bits, and you get a blinking reverse image. Versatile enough?

The overall blink rate must be under firmware control, because different panels have different response times. The 1/16 s blink runs faster than any panel I have available, while a 4 s blink should be glacial enough for a nearly frozen panel. You

can also vary the blink rate to attract attention. Because the blink signal changes only after an entire frame, all the dots will blink at the same rate… unless you get tricky and rewrite the RAM on the fly, a trick I'll mention again in Chapter 16.

All of this hocus pocus depends on dots on the LCD panel being just bits in PC memory. The **LCD Refresh RAM** appears in the PC's address space at D000:0000, just after the FDB's battery backed RAM at C800:0000 through C800:7FFF (a.k.a. C000:8000 through C000:FFFF). That circuit appeared in Chapter 7 and I discussed some of the issues involved in ISA bus memory accesses in Chapter 6.

The RAM Access Control Circuitry synchronizes **Dot Clock** with the ISA bus signals during each memory access, leaving the RAM in the proper state by the time each bus cycle finishes. Basically, the ISA bus runs slowly enough that we can pull a fast one on it! A 120 ns RAM cycles quickly enough for this application, so we don't need any particularly exotic hardware, either.

In desktop PCs, however, the 64 KB between D000:0000 and D000:FFFF may form an Expanded Memory page frame. Address space being the most precious commodity below the 1 MB line, EMS boards or EMM386 programs may use that 64 KB block as a window into megabytes of expanded RAM. You can move the **LCD Refresh RAM** to A000:0000 or B000:0000 if your system doesn't have a video card at one of those locations… not likely in a desktop system, I dare say.

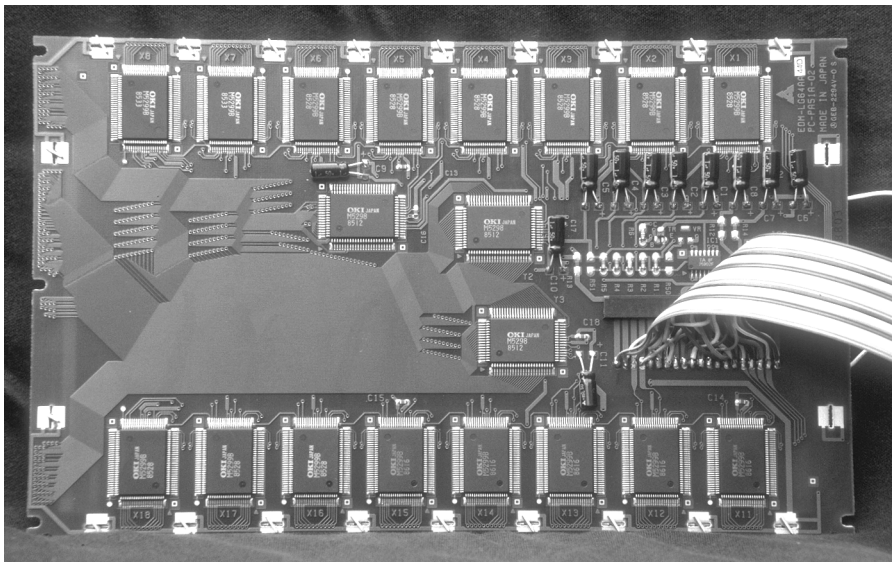
Variations on a Theme

If a 640×200 panel doesn't have enough dots for you, the next step up is 640×400. With VGA resolution at 640×480, SVGA at 800×600, and current laptops hitting the 8514/A level of 1024×768, everyone simply *must* maintain the pace. As a result, you can find older, smaller panels at reasonable prices. In fact, you could probably go into OEM production with surplus panels from previous-generation laptops.

Doubling the number of lines obviously doubles the number of dots visible on the panel. Because the overall refresh rate must remain about the same to avoid flicker, the panel must receive twice as many bits in about the same time. Panel designers have only two choices: transfer twice the number of bits per **Dot Clock** cycle, or send the same bits twice as fast by doubling the **Dot Clock** frequency. I have panels using each method and designed the Graphic LCD Interface for both.

Photo 2 shows the back of a Matsushita EDM LG64AA44D 640×400 panel, which, compared with Photo 1, has eight additional column driver chips. Each group of eight drivers handles four bits, so this panel receives eight bits in each **Dot Clock** cycle.

Photo 2
Compared to the 640x200 LCD in Photo 1, Matsushita's EDM LG64AA44D 640x400 LCD panel has eight additional column driver chips that accept four more bits per Dot Clock cycle. The white wires on the right carry AC power to the electroluminescent backlight.

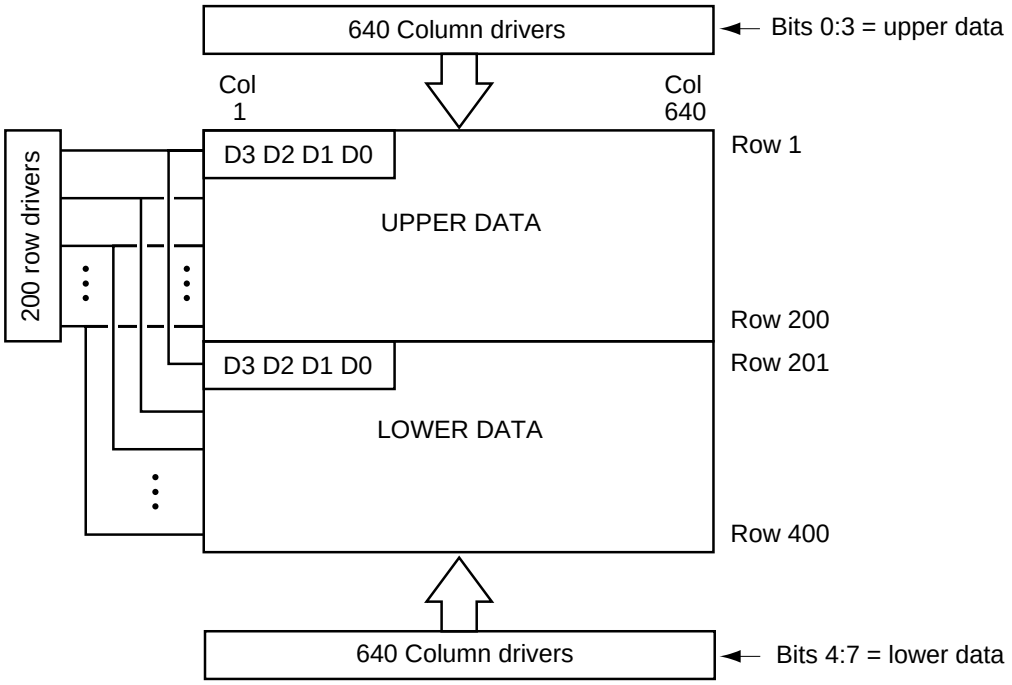


It has only three row driver chips, with each of the 200 outputs connected to *two* rows instead of one. Figure 8 sketches the layout: the display surface is split in two sections, each with separate column drivers. Thus, two separate rows are active simultaneously, with the two column driver sets presenting different data to the active row. You can think of the LG64AA44D as two 640×200 panels, butted together top-to-bottom on the same piece of glass.

Because the LG64AA44D requires eight bits on each cycle, the Graphic LCD Interface's data multiplexer serves no function. The Blinking and MUX Control logic emits a constant zero that routes the low-order nybble from the data latch through the MUX to the LCD connector. The high-order nybble goes directly from the latch to the connector and the panel latches all eight bits simultaneously on each **`Dot Clock`** cycle.

Bits 0:3 of each RAM byte appear on the upper half of the panel, while bits 4:7 appear on the lower half. The firmware must account for the fact that the bits in each byte show up at two widely separated locations. This is not the worst arrangement you'll meet on an LCD panel, as you'll see in a few pages.

Figure 8
Although the Matsushita LG64AA44D display has 400 rows of 640 dots, it uses only 200 row drivers. Each row driver activates two rows, one in each half of the display. One set of column drivers displays dots in the upper half, while an additional set drives the lower half. Because each set of column drivers handles four bits, the panel accepts eight bits on each 480 ns Dot Clock cycle.



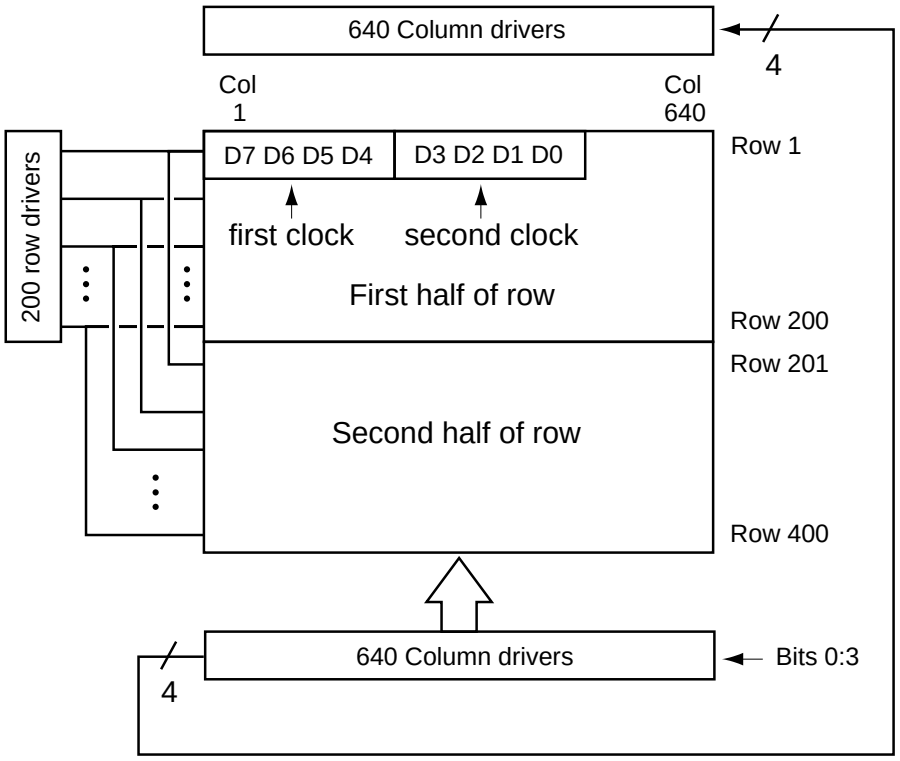
The additional dots in this display are basically free, because the Graphic LCD Interface already includes a byte-wide RAM. All it takes is a little firmware to plunk the dots in the right spots. In fact, the same code can handle either 200- or 400-line panels. Just don't blink that big display! (Go ahead, try it…)

Conversely, the Sharp LM64015T 640×400 panel diagrammed in Figure 9 uses the other technique: it accepts four bits on each double-speed 240 ns **`Dot Clock`** cycle. The two sets of column drivers run in series rather than parallel. In essence, we have the same chips as the LG64AA44D, but they run faster to keep up with the data. You'll recall that CMOS circuitry dissipates more power when it runs faster, which may affect the type of panel you select for a battery powered system.

In effect, you have a 1280×200 panel whacked in half, with the two 640×200 pieces stacked atop each other. Each logical row, twice the length of a visible row, requires 320 double-speed **`Dot Clocks`**. The second half of each row appears on the bottom of the panel.

The Blinking and MUX Control logic switches the multiplexer between the two nybbles on each half of the **`Dot Clock`** cycle. Because the panel accepts one nybble on each half cycle, the data bits show up together on the screen. It's easy to write graphic routines for a panel where bits 4:7 adjoin bits 0:3 in their natural order, even if the rows seem to be twice as long as they really are.

Figure 9
The Sharp LM64015T has 400 rows of 640 dots, with two sets of column shift registers chained together to hold 1280 bits. Each of the 200 row drivers activates two physical dot rows, but, unlike the Matsushita panel shown in Figure 8, they show two halves of a single logical row. Each double row requires 320 cycles of a 240 ns Dot Clock, transferring 1280 bits in 320 groups of four.The Toshiba TLY-365-121 mentioned in the text has a similar layout with 100 row drivers, 200 rows, and a 480 ns Dot Clock.



The Graphic LCD Interface RAM circuitry runs at the same speed as before, fetching and latching a new byte every 480 ns. The trick lies in the multiplexer, which must now present each nybble for 240 ns. The panel runs from a double-speed **`Dot Clock`**, but that doesn't affect the rest of the interface.

The LM64015T uses a blindingly bright, cold cathode, fluorescent tube backlight that requires about 1 kV to fire up. Although you can read some lower-performance electroluminescent panels when they're unlit, this one is *completely* useless without its backlight. I don't know of a consistent surplus source for the special CCFT inverters and a new inverter costs nearly as much as a surplus panel. But, it's a nice panel if you can get it working…

Along the same lines, the Toshiba TLY-365-121 640×200 panel has two sets of column driver chips and a pair of row driver chips chained in series. It clocks four bits every 480 ns, with each line using 320 **`Dot Clock`**s. This is also a 1280×100 panel in disguise, with the two halves of each row stacked atop each other.

The panel connections resemble Figure 9, with 100 row drivers simultaneously activating pairs of the 200 display rows. Although you can think of this as a double-speed version of a 960 ns 640×100 panel, I doubt one of those ever appeared, if only because the aspect ratio seems so weird. In any event, the Graphic LCD Interface handles this panel with ease.

My TLY-365-121 can serve as an example of what happens when you buy surplus stuff: it arrived with the wrong documentation. Mercifully, the connector pinout was nearly correct and I didn't fry the electronics while discovering the confusion. It seems the panel expects one variable negative supply on pin 14 rather than a fixed -22.5 V supply with a separate contrast control on pin 13. At least, that's the way it worked for me.

I've also looked at the Hitachi LM215, a 480×128 display sporting *four* sets of column drivers clocked at 960 ns. Each of the four data bits paints one quadrant and, as you can imagine, drawing anything becomes an intricate bit twiddling exercise. The Graphic LCD Interface can handle this one with an additional flipflop that doubles the 480 ns **`Dot Clock`** period. As you'll see, we must also store duplicate data in successive even/odd bytes to keep the output latch stable throughout the entire 960 ns cycle.

The Epson EG7004 640×200 panel requires an extra set of clock signals that route sync pulses through the shift registers. Think of that one as a challenge… you can find much easier panels to use, even as surplus items. Given the number of other panels in my collection, I gave up.

Release Notes

One thing is absolutely certain: bigger, better, and faster graphic LCD panels will appear for your use as the laptop computer market churns along. Somewhat to my surprise, I recently found a 640×480 panel for $9.95, *including* that usually impossible-to-find CCFT inverter.

After you acquire a few LCD panels, spend some time reading the documentation and thinking about how they work, before firing up your soldering iron and keyboard. Although the Graphic LCD hardware cannot handle every panel you'll find, you can probably adapt the basic design to cope with nearly anything. The main limitations will be the cramped address space available for the **`LCD Refresh RAM`** on the ISA bus and the higher refresh rates used by newer panels.

I'll discuss some modifications to the basic design in the next few chapters, as we explore the circuitry and firmware.