The Hitchhiker's Guide to X386/XFree86 Video Timing
(or, Tweaking your Monitor for Fun and Profit)

Eric S. Raymond *esr@snark.thyrsus.com* [1]

This is version 1.0, Jan 8th 1993.

Contents

1 Introduction

Please direct comments, criticism, and suggestions for improvement to *esr@snark.thyrsus.com*.

The XFree86 server allows users to configure their video subsystem and thus encourages best use of existing hardware. This tutorial is intended to help you learn how to generate your own timing numbers to make optimum use of your video card and monitor.

We'll present a method for getting something that works, and then show you how you can experiment starting from that base to develop settings that optimize for your taste.

If you already have a mode that almost works (in particular, if one of predefined VESA modes gives you a stable display but one that's displaced right or left, or too small, or too large) you can go straight to the section on Fixing Problems. This will enlighten you on ways to tweak the timing numbers to achieve particular effects.

XFree86 allows you to hot-key between different modes defined in XF86Config (see XF86Config.man for details). Use this capability to save yourself hassles! When you want to test a new mode, give it a unique mode label and add it to the *end* your hot-key list. Leave a known-good mode as the default to fall back on if the test mode doesn't work. The Xconfig section at the end of the second Example Calculation provides a good example of how to record your experiments in a way that will help you quickly converge on a solution.

First check out the `Monitors` file in `lib/X11/doc` If your monitor is in it, you can probably skip the rest of this document! You may need to scale some of the timing numbers if the clock used to generate the mode in the database doesn't match what your card has available, but that's easy.

2 How Video Displays Work

Knowing how the display works is essential to understanding what numbers to put in the various fields in the file Xconfig. Those values are used in the lowest levels of controlling the display by the XFree86 server.

The display generates a picture from a series of dots. The dots are arranged from left to right to form lines. The lines are arranged from top to bottom to form the picture. The dots emit light when they are struck by the electron beam inside the display. To make the beam strike each dot for an equal amount of time, the beam is swept across the display in a constant pattern.

The pattern starts at the top left of the screen, goes across the screen to the right in a straight line, and stops temporarily on the right side of the screen. Then the beam is swept back to the left side of the display, but down one line. The new line is swept from left to right just as the first line was. This pattern is repeated until the bottom line on the display has been swept. Then the beam is moved from the bottom right corner of the display to the top left corner, and the pattern is started over again.

Starting the beam at the top left of the display is called the beginning of a frame. The frame ends when the beam reaches the the top left corner again as it comes from the bottom right corner of the display. A frame is made up of all of the lines the beam traced from the top of the display to the bottom.

If the electron beam were on all of the time it was sweeping through the frame, all of the dots on the display would be illuminated. There would be no black border around the edges of the display. At the edges of the display the picture would become distorted because the beam is hard to control there. To reduce the distortion, the dots around the edges of the display are not illuminated by the beam even though the beam may be pointing at them. The viewable area of the display is reduced this way.

Another important thing to understand is what becomes of the beam when no spot is being painted on the visible area. The time the beam would have been illuminating the side borders of the display is used for sweeping the beam back from the right edge to the left and moving the beam down to the next line. The time the beam would have been illuminating the top and bottom borders of the display is used for moving the beam from the bottom-right corner of the display to the top-left corner.

The adapter card generates the signals which cause the display to turn on the electron beam at each dot to generate a picture. The card also controls when the display moves the beam from the right side to the left and down a line by generating a signal called the horizontal sync (for synchronization) pulse. One horizontal sync pulse occurs at the end of every line. The adapter also generates a vertical sync pulse which signals the display to move the beam to the top-left corner of the display. A vertical sync pulse is generated near the end of every frame.

The display requires that there be short time periods both before and after the horizontal and vertical sync pulses so that the position of the electron beam can stabilize. If the beam can't stabilize, the picture will not be steady.

In a later section, we'll come back to these basics with definitions, formulas and examples to help you use them.

3 Basic Things to Know about your Display and Adapter

There are some fundamental things you need to know before hacking an Xconfig entry. These are:

1. your monitor's horizontal and vertical sync frequency options
2. your video adapter's driving clock frequency, or "dot clock"
3. your monitor's bandwidth

The monitor sync frequencies:

The horizontal sync frequency are just the number of times per second the monitor can write a horizontal scan line; it is the single most important statistic about your monitor. The vertical sync frequency is the number of times per second the monitor can traverse its beam vertically.

Sync frequencies are usually listed on the specifications page of your monitor manual. The vertical sync frequency number is typically calibrated in Hz (cycles per second), the horizontal one in KHz (kilocycles per second). The usual ranges are between 50 and 80Hz vertical, and between 31 and 135KHz horizontal.

If you have a multisync monitor, these frequencies will be given as ranges. Some monitors, especially lower-end ones, have multiple fixed frequencies. These can be configured too, but your options will be severely limited by the built-in monitor characteristics. Choose the highest frequency pair for best resolution. And be careful — trying to clock a fixed-frequency monitor at a higher speed than it's designed for can damage it.

The card driving clock frequency:

Your video adapter manual's spec page will usually give you the card's dot clock (that is, the total number of pixels per second it can write to the screen). If you don't have this information, the X server will get it for you. Even if your X locks up your monitor, it will emit a line of clock and other info to standard output. If you redirect this to a file, it should be saved even if you have to reboot to get your console back.

If you're using SGCS X, the line will look something like the following example, collected from a Swan local-bus S3 adapter. XFree86 uses a slightly different multi-line format.

```
WGA: 86C911 (mem: 1024k clocks: 25 28 40 3 50 77 36 45 0 0 79 31 94 65 75 71)
---  ------       -----         -------------------------------------------
 |     |            |                 Possible driving frequencies in MHz
 |     |            +-- Size of on-board frame-buffer RAM
 |     +-- Chip type
 +-- Server type
```

Note: do this with your machine unloaded (if at all possible). Because X is an application, its timing loops can collide with disk activity, rendering the numbers above inaccurate. Do it several times and watch for the numbers to stabilize; if they don't, start killing processes until they do. SVr4 users: the mousemgr process is particularly likely to mess you up.

In order to avoid the clock-probe inaccuracy, you should clip out the clock timings and put them in your Xconfig as the value of the Clocks property — this suppresses the timing loop and gives X an exact list of the clock values it can try. Using the data from the example above:

```
wga
        Clocks  25 28 40 3 50 77 36 45 0 0 79 31 94 65 75 71
```

On systems with a highly variable load, this may help you avoid mysterious X startup failures. It's possible for X to come up, get its timings wrong due to system load, and then not be able to find a matching dot clock in its config database — or find the wrong one!

The monitor's video bandwidth:

Finally, it's useful to know your monitor's video bandwidth, so you know approximately what the highest dot clock you can use is. There's a lot of give here, though — some monitors can run as much as 30% over their nominal bandwidth.

Knowing the bandwidth will enable you to make more intelligent choices between possible configurations. It may affect your display's visual quality (esp. sharpness for fine details).

Your monitor's video bandwidth should be included on the manual's spec page. If it's not, look at the monitor's highest rated resolution. As a rule of thumb, here's how to translate these into bandwidth estimates (and thus into rough upper bounds for the dot clock you can use):

```
640x480                 25
800x600                 36
1024x768                65
1024x768 interlaced     45
1280x1024               110
```

BTW, there's nothing magic about this table; these numbers are just the lowest dot clocks per resolution in the standard XFree86 Modes database. The bandwidth of your monitor may be higher than the minimum needed for its top resolution, so don't be afraid to try a dot clock a few MHz higher.

Also note that bandwidth is seldom an issue for dot clocks under 65MHz or so. With an SVGA and most hi-res monitors, you can't get anywhere near the limit of your monitor's video bandwidth. The following are examples:

```
Brand                           Video Bandwidth
----------                      ---------------
NEC 4D                          75Mhz
Nano 907a                       50Mhz
Nano 9080i                      60Mhz
Mitsubishi HL6615               110Mhz
Mitsubishi Diamond Scan         100Mhz
IDEK MF-5117                    65Mhz
IOCOMM Thinksync-17 CM-7126     136Mhz
HP D1188A                       100Mhz
Philips SC-17AS                 110Mhz
Swan SW617                      85Mhz
```

Even low-end monitors usually aren't terribly bandwidth-constrained for their rated resolutions. The NEC Multisync II makes a good example — it can't even display 800x600 per its spec. It can only display 800x560. For such low resolutions you don't need high dot clocks or a lot of bandwidth; probably the best you can do is 32Mhz or 36Mhz, both of them are still not too far from the monitor's rated video bandwidth of 30Mhz.

At these two driving frequencies, your screen image may not be as sharp as it should be, but definitely of tolerable quality. Of course it would be nicer if NEC Multisync II had a video bandwidth higher than, say, 36Mhz. But this is not critical for common tasks like text editing, as long as the difference is not so significant as to cause severe image distortion (your eyes would tell you right away if this were so).

What these control:

The sync frequency ranges of your monitor, together with your video adapter's dot clock, determine the ultimate resolution that you can use. But it's up to the driver to tap the potential of your hardware. A superior hardware combination without an equally competent device driver is a waste of money. On the other hand, with a versatile device driver but less capable hardware, you can push the hardware's envelope a little. This is the design philosophy of XFree86.

4 Interpreting the Basic Specifications

This section explains what the specifications above mean, and some other things you'll need to know. First, some definitions. Next to each in parens is the variable name we'll use for it when doing calculations

horizontal sync frequency (HSF)

Horizontal scans per second (see above).

vertical sync frequency (VSF)

Vertical scans per second (see above). Mainly important as the upper limit on your refresh rate.

dot clock (DCF)

More formally, 'driving clock frequency'; sometimes loosely called 'bandwidth'. The frequency of the crystal or VCO on your adaptor — the maximum dots-per-second it can emit.

video bandwidth (VB)

The highest frequency at which your monitor's video signal can change. This constrains the highest dot clock you can use and the overall sharpness of fine details in the video image.

frame length (HFL, VFL)

Horizontal frame length (HFL) is the number of dot-clock ticks needed for your monitor's electron gun to scan one horizontal line, *including the inactive left and right borders*. Vertical frame length (VFL) is the number of scan lines in the *entire* image, including the inactive top and bottom borders.

screen refresh rate (RR)

The number of times per second your screen is repainted. Higher frequencies are better, as they reduce flicker. 60Hz is good, VESA-standard 72Hz is better. Compute it as

$$RR = DCF/(HFL * VFL)$$

Note that the product in the denominator is *not* the same as the monitor's visible resolution, but typically somewhat larger. We'll get to the details of this below.

About Bandwidth:

Monitor makers like to advertise high bandwidth because it constrains the sharpness of intensity and color changes on the screen. A high bandwidth means smaller visible details.

Your monitor uses electronic signals to present an image to your eyes. Such signals always come in in wave form once they are converted into analog form from digitized form. They can be considered as combinations of many simpler wave forms each one of which has a fixed frequency, many of them are in the Mhz range, eg, 20Mhz, 40Mhz, or even 70Mhz. Your monitor video bandwidth is, effectively, the highest-frequency analog signal it can handle without distortion.

For our purposes, bandwidth is mainly important as an approximate cutoff point for the highest dot clock you can use.

Sync Frequencies snd the Refresh Rate:

Each horizontal scan line on the display is just the visible portion of a frame-length scan. At any instant there is actually only one dot active on the screen, but with a fast enough refresh rate your eye's persistence of vision enables you to "see" the whole image.

Here are some pictures to help:

```
-----------------------
|                       |     The horizontal frame length
|->->->->->->->->->->-> |     is the time in dot clocks
|                      )|     required for the
|<-----<-----<-----<--- |     electron beam to trace
|                       |     a pattern like this
|                       |
|                       |
|                       |
|_______________________|
```

```
-----------------------
|        ^              |     The vertical frame length
|       ^ |             |     is the time in dot clocks
|       | v             |     required for the
|       ^ |             |     electron beam to trace
|       | |             |     a pattern like this
|       ^ |             |
|       | v             |
|       ^ |             |
|_______|_v_____________|
```

Remember that the actual raster scan is a very tight zigzag pattern; that is, the beam moves left ¡-¿ right and at the same time up ¡-¿ down.

Now we can see how the dot clock and frame size relates to refresh rate. By definition, one hertz (hz) is one cycle per second. So, if your horizontal frame length is HFL and your vertical frame length is VFL, then to cover the entire screen takes (HFL * VFL) ticks. Since your card emits DCF ticks per second by definition, then obviously your monitor's electron gun(s) can sweep the screen from left to right and back and from bottom to top and back DCF / (HFL * VFL) times/sec. This is your screen's refresh rate, because it's how many times your screen can be updated thus REFRESHED per second!

You need to understand this concept to design a configuration which trades off resolution against flicker in whatever way suits your needs.

5 Tradeoffs in Configuring your System

Another way to look at the formula we derived above is

DCF = RR * HFL * VFL

That is, your dot clock is fixed. You can use those dots per second to buy either refresh rate, horizontal resolution, or vertical resolution. If one of those increases, one or both of the others must decrease.

Note, though, that your refresh rate cannot be greater than the maximum vertical sync frequency of your monitor. Thus, for any given monitor at a given dot clock, there is a minimum product of frame lengths below which you can't force it.

In choosing your settings, remember: if you set RR too low, you will get mugged by screen flicker.

You probably do not want to pull your refresh rate below 60Hz. This is the flicker rate of fluorescent lights; if you're sensitive to those, you need to hang with 72MHz, the VESA ergonomic standard.

Flicker is very eye-fatiguing, though human eyes are adaptable and peoples' tolerance for it varies widely. If you face your monitor at a 90% viewing angle, are using a dark background and a good contrasting color for foreground, and stick with low to medium intensity, you *may* be comfortable at as little as 45Hz.

The acid test is this: open a xterm with pure white back-ground and black foreground using xterm -bg white -fg black and make it so large as to cover the entire viewable area. Now turn your monitor's intensity to 3/4 of its maximum setting, and turn your face away from the monitor. Try peeking at your monitor sideways (bringing the more sensitive peripheral-vision cells into play). If you don't sense any flicker or if you feel the flickering is tolerable, then that refresh rate is fine with you. Otherwise you better configure a higher refresh rate, because that semi-invisible flicker is going to fatigue your eyes like crazy and give you headaches, even if the screen looks OK to normal vision.

So let's say you've picked a minimum acceptable refresh rate. In choosing your HFL and VFL, you'll have some room for maneuver.

6 Memory Requirements

Available frame-buffer RAM may limit the resolution you can achieve on color or gray-scale displays. It probably isn't a factor on displays that have only two colors, white and black with no shades of gray in between.

For 256-color displays, a byte of video memory is required for each visible dot to be shown. This byte contains the information that determines what mix of red, green, and blue is generated for its dot. To get the amount of memory required, multiply the number of visible dots per line by the number of visible lines. For a display with a resolution of 800x600, this would be 800 x 600 = 480,000, which is the number of visible dots on the display. This is also, at one byte per dot, the number of bytes of video memory that are necessary on your adapter card.

Thus, your memory requirement will typically be (HR * VR)/1024 Kbytes of VRAM, rounded up. In the example case, we'd need (936 * 702)/1024 = 642K. So if you have one meg, you'll have extra for virtual-screen panning.

However, if you only have 512K on board, then you can't use this resolution. Even if you have a good monitor, without enough video ram, you can't take advantage of your monitor's potential. On the other hand, if your SVGA has one meg, but your monitor can display at most 800x600, then high resolution is beyond your reach anyway.

Don't worry if you have more memory than required; XFree86 will make use of it by allowing you to scroll your viewable area (see the Xconfig file documentation on the virtual screen size parameter). Remember also that a card with 512K bytes of memory really doesn't have 512,000 bytes installed, it has 512 x 1024 = 524,288 bytes.

If you're running SGCS X using an S3 card, and are willing to live with 16 colors (4 bits per pixel), you can set depth 4 in Xconfig and effectively double the resolution your card can handle. S3 cards, for example, normally do 1024x768x256. You can make them do 1280x1024x16 with depth 4.

7 Computing Frame Sizes

Warning: this method was developed for multisync monitors. It will probably work with fixed-frequency monitors as well, but no guarantees!

Start by dividing DCF by your highest available HSF to get the number of horizontal sweeps per second available.

For example; suppose you have a Sigma Legend SVGA with a 65MHz dot clock, and your monitor has a 55KHz horizontal scan frequency. The quantity (DCF / HSF) is then 1181.

Now for our first bit of black magic. You need to round this figure to the nearest multiple of 8. This has to do with the VGA hardware controller used by SVGA and S3 cards; it uses an 8-bit register, left-shifted 3 bits, for what's really an 11-bit quantity. Other card types such as ATI 8514/A may not have this requirement, but we don't know and the correction can't hurt. So round the usable horizontal scans per second figure to 1176.

This figure (DCF / HSF rounded to a multiple of 8) is the minimum HFL you can use. You can get longer HFLs (and thus, possibly, more horizontal dots on the screen) by setting the sync pulse to produce a lower HSF. But you'll pay with a slower and more visible flicker rate.

As a rule of thumb, 80% of the horizontal frame length is available for horizontal resolution, the visible part of the horizontal scan line (this allows, roughly, for borders and sweepback time – that is, the time required for the beam to move from the right screen edge to the left edge of the next raster line). In this example, that's 944 ticks.

Now, to get the normal 4:3 screen aspect ratio, set your vertical resolution to 3/4ths of the horizontal resolution you just calculated. For this example, that's 708 ticks. To get your actual VFL, multiply that by 1.05 to get 743 ticks.

About that 4:3 — a ratio of 4:3 for width to height of the displayed area approximates the Golden Section, (1 + sqrt(5))/2. Human beings seem to be wired to find this kind of rectangle pleasant to look at; accordingly, video tubes and the standard resolutions such as 800x600, 640x480 and 1024x768 all approximate it. Though it's psychologically magic, it's not technically magic; nothing prevents you from using a non-Golden-Section ratio if that will get the best use out of your screen real estate.

So, HFL=1176 and VFL=743. Dividing 65MHz by the product of the two gives us a nice, healthy 74.4Hz refresh rate. Excellent! Better than VESA standard! And you got 944x708 to boot, more than the 800 by 600 you were probably expecting. Not bad at all!

You can even improve the refresh rate further, to almost 76 Hz, by using the fact that monitors can often sync horizontally at 2khz or so higher than rated, and by lowering VFL somewhat (that is, taking less than 75% of 944 in the example above). But before you try this "overdriving" maneuver, if you do, make *sure* that your monitor electron guns can sync up to 76 Hz vertical. (the popular NEC 4D, for instance, cannot. It goes only up to 75 Hz VSF).

So far, most of this is simple arithmetic and basic facts about raster displays. Hardly any black magic at all!

8 Black Magic and Sync Pulses

OK, now you've computed HFL/VFL numbers for your chosen dot clock, found the refresh rate acceptable, and checked that you have enough VRAM. Now for the real black magic – you need to know when and where to place synchronization pulses.

The sync pulses actually control the horizontal and vertical scan frequencies of the monitor. The HSF and VSF you've pulled off the spec sheet are nominal, approximate maximum sync frequencies. The sync pulse in the signal from the adapter card tells the monitor how fast to actually run.

Recall the two pictures above? Only part of the time required for raster-scanning a frame is used for displaying viewable image (ie. your resolution).

Horizontal Sync:

By previous definition, it takes HFL ticks to trace the a horizontal scan line. Let's call the visible tick count (your horizontal screen resolution) HR. Then Obviously, HR ¡ HFL by definition. For concreteness, let's assume both start at the same instant as shown below:

```
|--- -- -- -- -- -- -- -- -- -- -- -- --
|_ _ _ _ _ _ _ _ _ _ _ _                |
|_______________________|_______________|_____
0                       ^               ^     unit: ticks
                        |   ^       ^   |
                        HR  |       |  HFL
                        |   |<----->|   |
                        |<->|  HSP  |<->|
                        HGT1        HGT2
```

Now, we would like to place a sync pulse of length HSP as shown above, ie, between the end of clock ticks for display data and the end of clock ticks for the entire frame. Why so? because if we can achieve this, then your screen image won't shift to the right or to the left. It will be where it supposed to be on the screen, covering squarely the monitor's viewable area.

Furthermore, we want about 30 ticks of "guard time" on either side of the sync pulse. This is represented by HGT1 and HGT2. In a typical configuration HGT1 != HGT2, but if you're building a configuration from scratch, you want to start your experimentation with them equal (that is, with the sync pulse centered).

The symptom of a misplaced sync pulse is that the image is displaced on the screen, with one border excessively wide and the other side of the image wrapped around the screen edge, producing a white edge line and a band of "ghost image" on that side. A way-out-of-place vertical sync pulse can actually cause the image to roll like a TV with a mis-adjusted vertical hold (in fact, it's the same phenomenon at work).

If you're lucky, your monitor's sync pulse widths will be documented on its specification page. If not, here's where the real black magic starts...

You'll have to do a little trial and error for this part. But most of the time, we can safely assume that a sync pulse is about 3.5 to 4.0 microsecond in length.

For concreteness again, let's take HSP to be 3.8 microseconds (which btw, is not a bad value to start with when experimenting).

Now, using the 65Mhz clock timing above, we know HSP is equivalent to 247 clock ticks (= 65x10**6 * 3.8 *10**(-6)) $recallM = 10 * *6, micro = 10 * *(-6)$

Vertical Sync:

Going back to the picture above, how do we place the 247 clock ticks as shown in the picture?

Using our example, HR is 944 and HFL is 1176. The difference between the two is 1176-944=232 ¡ 247! Obviously we have to do some adjustment here. What can we do?

The first thing is to raise 1176 to 1184, and lower 944 to 936. Now the difference = 1184-936= 248. Hmm, closer.

Next, instead using 3.8, we use 3.5 for calculating HSP; then, we have 65*3.5=227. Looks better. But 248 is not much higher than 227. It's normally necessary to have 30 or so clock ticks between HR and the start of SP, and the same for the end of SP and HFL. AND they have to be multiple of eight! Are we stuck?

No! let's do this, 936% 8==0, (936+32)% 8==0 too. But 936+32=968, 968+227=1195, 1195+32=1227. Hmm.. this looks not too bad. But it's not a multiple of 8, so lets round it up to 1232.

But now we have potential trouble, the sync pulse is no longer placed right in the middle between h and H any more. Happily, using our calculator we find 1232-32=1200 is also a multiple of 8 and (1232-32)-968=232 corresponding using a sync pulse of 3.57 micro second long, still reasonable.

In addition, 936/1232 0.76 or 76%, still not far from 80%, so it should be all right.

Furthermore, using the current horizontal frame length, we basically ask our monitor to sync at 52.7khz(=65Mhz/1232) which is within its capability. No problems.

Using rules of thumb we mentioned before, 936*75%=702, This is our new vertical resolution. 702*1.05=737, our new vertical frame length.

Screen refresh rate = 65Mhz/(737*1232)=71.6 Hz. This is still excellent.

Figuring the vertical sync pulse layout is similar:

```
|___ -- -- -- -- -- -- -- -- -- -- -- --
|_ _ _ _ _ _ _ _ _ _ _ _               |
|_______________________|_______________|_____
0                      VR              VFL     unit: ticks
                        ^    ^       ^
                        |    |       |
                        |<->|<----->|
                         VGT    VSP
```

We start the sync pulse just past the end of the vertical display data ticks. VGT is the vertical guard time required for the sync pulse. Most monitors are comfortable with a VGT of 0 (no guard time) and we'll use that in this example. A few need two or three ticks of guard time, and it usually doesn't hurt to add that.

Returning to the example: since by the definition of frame length, a vertical tick is the time for tracing a complete HORIZONTAL frame, therefore in our example, it is 1232/65Mhz=18.95us.

Experience shows that a vertical sync pulse should be in the range of 50us and 300us. As an example let's use 150us, which translates into 8 vertical clock ticks (150us/18.95us 8).

9 Putting it All Together

The Xconfig file Table of Video Modes contains lines of numbers, with each line being a complete specification for one mode of X-server operation. The fields are grouped into four sections, the name section, the clock frequency section, the horizontal section, and the vertical section.

The name section contains one field, the name of the video mode specified by the rest of the line. This name is referred to on the "Modes" line of the Graphics Driver Setup section of the Xconfig file. The name field may be omitted if the name of a previous line is the same as the current line.

The dot clock section contains only the dot clock (what we've called DCF) field of the video mode line. The number in this field specifies what dot clock was used to generate the numbers in the following sections.

The horizontal section consists of four fields which specify how each horizontal line on the display is to be generated. The first field of the section contains the number of dots per line which will be illuminated to form the picture (what we've called HR). The second field of the section indicates at which dot the horizontal sync pulse will begin. The third field indicates at which dot the horizontal sync pulse will end. The fourth field specifies the total horizontal frame length (HFL).

The vertical section also contains four fields. The first field contains the number of visible lines which will appear on the display (VR). The second field indicates the line number at which the vertical sync pulse will begin. The third field specifies the line number at which the vertical sync pulse will end. The fourth field contains the total vertical frame length (VFL).

Example:

```
#Modename    clock  horizontal timing  vertical timing

"752x564"     40    752 784  944 1088  564 567 569 611
              44.5  752 792  976 1240  564 567 570 600
```

(Note: stock X11R5 doesn't support fractional dot clocks.)

For Xconfig, all of the numbers just mentioned - the number of illuminated dots on the line, the number of dots separating the illuminated dots from the beginning of the sync pulse, the number of dots representing the duration of the pulse, and the number of dots after the end of the sync pulse - are added to produce the number of dots per line. The number of horizontal dots must be evenly divisible by eight.

Example:

horizontal numbers: 800 864 1024 1088

The sample line has the number of illuminated dots (800) followed by the number of the dot when the sync pulse starts (864), followed by the number of the dot when the sync pulse ends (1024), followed by the number of the last dot on the horizontal line (1088).

Note again that all of the horizontal numbers (800, 864, 1024, and 1088) are divisible by eight! This is not required of the vertical numbers.

The number of lines from the top of the display to the bottom form the frame. The basic timing signal for a frame is the line. A number of lines will contain the picture. After the last illuminated line has been displayed, a delay of a number of lines will occur before the vertical sync pulse is generated. Then the sync pulse will last for a few lines, and finally the last lines in the frame, the delay required after the pulse, will be generated. The numbers that specify this mode of operation are entered in a manner similar to the following example.

Example:

vertical numbers: 600 603 609 630

This example indicates that there are 600 visible lines on the display, that the vertical sync pulse starts with the 603rd line and ends with the 609th, and that there are 630 total lines being used.

Note that the vertical numbers don't have to be divisible by eight!

Let's return to the example we've been working. According to the above, all we need to do from now on is to write our result into Xconfig as follows:

```
{$<$} name {$>$}   DCF     HR  SH1 SH2   HFL   VR  SV1 SV2 VFL
```

where SH1 is the start tick of the horizontal sync pulse and SH2 is its end tick; similarly, SV1 is the start tick of the vertical sync pulse and SV2 is its end tick.

```
#name    clock   horizontal timing   vertical timing    flag
936x702  65      936 968 1200 1232   702 702 710 737
```

No special flag necessary; this is a non-interlaced mode. Now we are really done.

10 Questions and Answers

Q. The example you gave is not a standard screen size, can I use it?

A. Why not? There is NO reason whatsoever why you have to use 640x480, 800x600, or even 1024x768. XFree86 driver lets you config your hardware with a lot of freedom. It usually takes two to three minutes to come up the right one. The important thing to shoot for is high refresh rate with reasonable viewing area. not high resolution at the price of eye-tearing flicker!

Q. It this the *only* resolution given the 65Mhz dot clock and 55Khz HSF?

A. Absolutely not! You are encouraged to follow the general procedure and do some trial-and-error to come up with a setting that's really to your liking. Experimenting with this can be lots of fun. Most settings may just give you nasty video hash, but nothing you do can actually damage a multi-sync monitor (unless you somehow force your card to clock it at way above its bandwidth — if you stick reasonably close to the highest resolution the monitor is documented to support this can't happen). Beware fixed-frequency monitors! This kind of hacking around *can* damage them.

Q. You just mentioned two standard resolutions. In Xconfig, there are many standard resolutions available, can you tell me whether there's any point in tinkering with timings?

A. Absolutely! Take, for example, the "standard" 640x480 listed in the current Xconfig. It employs 25Mhz driving frequency, frame lengths are 800 and 525 =¿ refresh rate 59.5Hz. Not too bad. But 28Mhz is a commonly available driving frequency from many SVGA boards. If we use it to drive 640x480, following the procedure we discussed above, you would get frame lengths like 812 and 505. Now the refresh rate is raised to 68Hz, a quite significant improvement over the standard one.

Q. But how about interlace/non-interlace?

A. At a fixed dot clock, an interlaced display is going to flicker worse than a non-interlaced one, which is why the market has moved away from them. What you buy with the increased flicker is higher resolution with a slower dot clock. If the DCF were fast enough (say 90MHz or up) even interlacing wouldn't produce flicker – but at present speeds, interlaced monitors are a bad idea for X.

Q. Can you summarize what we have discussed so far?

A. In a nutshell:

1. for any fixed driving frequency, raising max resolution incurs the penalty of lowering refresh rate and thus introducing more flicker.
2. if high resolution is desirable and your monitor supports it, try to get a SVGA card that provides a matching dot clock or DCF. The higher, the better!

11 Two More Example Calculations

Here's another hypothetical:

An adapter card has a 40 MHz clock rate. A display has a range of horizontal sync rates from 30 KHz to 37 KHz. The minimum number of dots per line is 40,000,000/37,000 = 1,081.081, or approximately 1,081 dots per line.

We'll use this number of dots per line in the following calculations. So each line on our display must have at least 1081 dots. We round this up to 1088 to make it divisible evenly by eight. Now let's assume that the horizontal sync pulse should be 3.8 microseconds long. We need to find out how many dots it takes to make a 3.8 microsecond pulse. We do this by first finding out how many microseconds are in one dot. Since there are 40,000,000 dots per second, 1/40,000,000 is the number of seconds per dot.

1/40,000,000 = .000000025 = .025 microseconds per dot

Thus the number of dots for a 3.8 microsecond sync pulse is

3.8 microseconds = D dots x .025 microseconds/dot

or

D dots = (3.8 microseconds) / (.025 microseconds/dot) = 152 dots

So we have 1088 dots per line, 800 of which are the illuminated ones, with 152 for the sync pulse. (Note that 152 is evenly divisible by eight. If it weren't we would round it up until it was evenly divisible.) This leaves us the task of calculating the time before and after the sync pulse that is necessary for the display.

The rule of thumb for this is that we need about 30 ticks of guard time. In this particular case, allocating 32 dots is convenient, because all the other quantities are divisible by 8. This results in the timing being 800 dots for the viewable area, 152 dots for the pulse, and 1088 - (800 + 152) = 136 dots to divide between the two other times. Half of 136 is 68 dots, so 68 dots are placed between the illuminated dots and the sync pulse, and 68 dots are placed after the sync pulse. The horizontal numbers in the Xconfig line then become

800 (800+68) (800+68+152) (800+68+152+68)

or

800 868 1020 1088

Now we want to calculate the vertical numbers. To begin, we must remember that the vertical numbers are not in terms of dots or microseconds per dot, but are expressed as numbers of lines! So we have to calculate how much time it takes to display a single line. That's easy, because we know each line is 1088 dots and each dot is .025 microsecond. Each line is, therefore,

(1088 dots/line) x (.025 microseconds/dot) = 27.2 microseconds/line

Since we chose 800 visible dots per line, let's choose the number of lines to be such that the ratio of horizontal to vertical is 4 to 3. Thus, 800 is 4 x 200, so the number of visible lines should be 3 x 200 = 600. Our target resolution is 800x600.

We know that a vertical sync pulse should be in the range of 50 to 300 microseconds. If we chose 150 microseconds as a typical sync pulse, we find how many lines 150 microseconds is by dividing 150 by 27.2 microseconds per line.

(150 microseconds/pulse) / (27.2 microseconds/line) = 5.51 lines/pulse

By rounding up (never down) to 6 lines/pulse we now have the vertical sync pulse width.

To guess at the total number of lines per frame (illuminated lines plus nonilluminated lines in the border) we assume (from "Videotiming...") that the total number of lines will be 5% more than the number of viewable lines. So the total number of lines is

(600 lines) x (1.05) = 630 total lines per frame

So now we must place the pulse in the time between the end of the illuminated lines and the end of the frame. Since we have 630 total lines, 600 illuminated lines, and 6 lines for the pulse, we have

630 - 600 - 6 = 24 lines left

Some displays don't mind if the pulse begins immediately after the illuminated lines, but others might want a line or two between the last illuminated line and the beginning of the sync pulse. Taking the latter course just to be safe, we add three lines between the last illuminated line and the beginning of the pulse. The rest of the lines are added after the pulse ends. So the vertical timing numbers become

600 (600+3) (600+3+6) (600+3+6+21)

or

600 603 609 630

Before we do anything else, we must check that the display can handle 630 lines/frame at 27.2 microseconds/line. We do this by calculating how many frames per second our configuration will generate, and comparing it to the display manual's entry for vertical sync rate. For 630 lines/frame at 27.2 micro- seconds/line, we have 630 x 27.2 = 17,136 microseconds/frame. 17,136 microseconds/frame is 0.017136 seconds/frame, or 1/0.017892 frames/second.

1 / (0.017136 seconds/frame) = 58.4 frames/second

If the manual says the vertical sync rate is 58.4 Hz, or 58.4 Hz is in the range of the display's vertical sync rate, we are fine. If the display cannot handle this rate, we'll have to change the number of lines per frame by adjusting all of the timings proportionally.

Now we combine the horizontal and vertical timing numbers together with the resolution and clock values to produce a test configuration for Xconfig. Our line becomes

"800x600" 40 800 868 1020 1088 600 603 609 630

Now we have a configuration of XFree86 to try. It may not work if any of our assumptions were grossly wrong, but in most cases it should at least give us a stable display. Now it takes a little experimentation to produce something pleasing.

An actual calculation

My adapter card has a 40 MHz crystal on it so I started with a 40 MHz clock rate. My display's maximum horizontal sync rate is 37 KHz, so the minimum dots per line are 40,000,000/37,000 = 1081. My display's vertical sync rate is the range from 50 Hz to 90 Hz.

My display's manual says that the largest horizontal sync pulse is 3.92 microseconds. With 0.025 microseconds per dot, the pulse is

(3.92 microseconds) / (.025 microseconds/dot) = 156.8 dots

Rounding this up to the nearest number evenly divisible by eight gives 160 dots.

The manual also says that the time between the last illuminated dot and the beginning of the sync pulse must be at least 0.67 microseconds. The number of dots in 0.67 microseconds at a 40 MHz clock rate - remember 40 MHz is .025 microseconds/dot - is

D dots = (0.67 microseconds) / (.025 microseconds/dot) = 26.8 dots

Since 26.8 is not evenly divisible by eight, round it up to 32 dots.

My display's manual says the time after the sync pulse should be 3.56 microseconds or more. In dots, 3.56 microseconds is

D dots = (3.56 microseconds) / (.025 microseconds/dot) = 142.4 dots

Round 142.4 up to 144, so that it's evenly divisible by eight.

So now for a horizontal line we have 800 illuminated dots, 32 dots between the illuminated dots and the sync pulse, 152 dots for the sync pulse, and 144 dots after the sync pulse.

800 + 32 + 160 + 144 = 1136

We now have a line that is 1136 dots long. This is greater than the 1088 we previously calculated, but remember that 1088 was the MINIMUM number of dots that could be on a line. So 1136 dots per line is okay for starters.

The numbers to enter on the Xconfig line so far are

"800x?" 40 800 (800+32) (800+32+160) (800+32+160+144)...

or

"800x?" 40 800 832 992 1136...

A line of 1136 dots at .025 microsecond/dot means that a line represents 1136 x .025 = 28.4 microseconds.

Since we chose 800 dots/line horizontal resolution, we choose 600 lines/frame as the vertical resolution.

My display's manual says that the vertical sync pulse must be at least 64 microseconds long. In terms of lines, 64 microseconds is

(64 microseconds/pulse) / (28.4 microseconds/line) = 2.25 lines/pulse

We round 2.25 up to 3 lines for the vertical sync pulse.

The manual says the time between the last displayed line and the start of the sync pulse must be at least 318 microseconds, and the delay after the end of the pulse must be at least 630 microseconds. We calculate how many lines each of these time periods represents as follows.

(318 microseconds) / (28.4 microseconds/line) = 11.20 lines

(630 microseconds) / (28.4 microseconds/line) = 22.18 lines

We round each of the times up to become 12 lines before the sync pulse and 23 lines after the pulse. This makes our vertical timing numbers

600 (600+12) (600+12+3) (600+12+3+23)

or

600 612 615 638

Checking the frame rate to see if it falls within the rate of the display, we see that 638 lines/frame at 28.4 microseconds/line is 18,119 microseconds/frame, which is 55.19 frames/second. My display can handle anything from 50 Hz to 90 Hz, so the timing is all right.

Putting the resolution, clock, horizontal, vertical timing numbers together on a video mode line in Xconfig results in

"800x600" 40 800 832 992 1136 600 612 615 638

This was the first video mode I tried. It turned out not to be very satisfactory because there was too much flicker. I tried other timings both above and below this setting as shown in the following example. I finally settled on the "784x614" mode as a compromise between flicker and resolution.

You'll notice that almost all of the clock frequencies are 40 MHz. Through experimentation I found that higher frequencies were beyond my adapter card's capabilities, and that lower frequencies didn't provide the resolution I wanted.

Example: Timings I have tried:

```
# the following line works but is right of center
"752x564"  40  752  784  944 1088  564 567 569 611
#        44.5  752 792  976 1240  564 567 570 600
#
# this line fixes the problem with the previous line
#"752x564" 40  752  816  976 1088  564 567 569 611
#
# trying to increase the vertical display size, it works
#"752x614" 40  752  816  976 1088  614 617 619 661
#
# trying to increase the horiz. display size, it works
#"784x564" 40  784  816  976 1088  564 567 569 611
#
# the following works but is to the right of center
#"784x614" 40  784  816  976 1088  614 617 619 661
#
# the following corrects the uncentered problem of the previous one
"784x614"  40  784  848 1008 1088  614 617 619 661
#
# trying to increase the display size
# the following works, the display is slightly off center to the left
#"800x614" 40  800  864 1024 1088  614 617 619 661
#
# the following corrects the problem of the previous entry
"800x614"  40  800  864 1024 1104  614 617 619 661
#
# increase the display size, it works
"816x614"  40  816  880 1040 1120  614 617 619 661
#
# increase the display size, it works
"800x620"  40  800  864 1024 1104  620 623 625 661
#
# increase the display size, it works
"816x620"  40  816  880 1040 1120  620 623 625 661
#
# increase the display size, it works
"832x630"  40  832  896 1056 1136  630 633 635 661
#
# change the display size, it works but flickers badly
"848x618"  40  848  912 1072 1152  618 621 623 661
```

12 Fixing Problems with the Image.

OK, so you've got your X configuration numbers. You put them in Xconfig with a test mode label. You fire up X, hot-key to the new mode, ... and the image doesn't look right. What do you do? Here's a list of common problems and how to fix them.

You *move* the image by changing the sync pulse timing. You *scale* it by changing the frame length (you need to move the sync pulse to keep it in the same relative position, otherwise scaling will move the image as well). Here are some more specific recipes:

The horizontal and vertical positions are independent. That is, moving the image horizontally doesn't affect placement vertically, or vice-versa. However, the same is not quite true of scaling. While changing the horizontal size does nothing to the vertical size or vice versa, the total change in both may be limited. In particular, if your image is too large in both dimensions you will probably have to go to a higher dot clock to fix it. Since this raises the usable resolution, it is seldom a problem!

The image is displaced to the left or right

To fix this, move the horizontal sync pulse. That is, increment or decrement (by a multiple of 8) the middle two numbers of the horizontal timing section that define the leading and trailing edge of the horizontal sync pulse.

If the image is shifted left (right border too large, you want to move the image to the right) decrement the numbers. If the image is shifted right (left border too large, you want it to move left) increment the sync pulse.

The image is displaced up or down

To fix this, move the vertical sync pulse. That is, increment or decrement the middle two numbers of the vertical timing section that define the leading and trailing edge of the vertical sync pulse.

If the image is shifted up (lower border too large, you want to move the image down) decrement the numbers. If the image is shifted down (top border too large, you want it to move up) increment the numbers.

The image is too wide (too narrow) horizontally

To fix this, increase (decrease) the horizontal frame length. That is, change the fourth number in the first timing section. To avoid moving the image, also move the sync pulse (second and third numbers) half as far, to keep it in the same relative position.

The image is too deep (too shallow) vertically

To fix this, decrease (increase) the vertical frame length. That is, change the fourth number in the second timing section. To avoid moving the image, also move the sync pulse (second and third numbers) half as far, to keep it in the same relative position.

Any distortion that can't be handled by combining these techniques is probably evidence of something more basically wrong, like a calculation mistake or a faster dot clock than the monitor can handle.

Finally, remember that increasing either frame length will decrease your refresh rate, and vice-versa.

```
$XConsortium: VidModes.sgml,v 1.3 95/01/27 16:14:33 kaleb Exp $
$XFree86: xc/programs/Xserver/hw/xfree86/doc/sgml/VidModes.sgml,v 3.5 1995/01/28 16:02:40 dawes Exp $
```