Video Signals And Timing

From OSDev Wiki

WARNING: Improperly changing CRTC or hardware settings can be harmful to the video card and attached monitor. Configuring graphics hardware may be a rewarding process, and many have achieved their goals with success and have even tampered with them well beyond their original specifications. Fact remains that there are *several* known instances of monitors - CRT and LCD alike - burning out after being fed poor data. Similarly, the video card might lack similar safeguards against accidental misconfiguration.

DISCLAIMER: The information provided might not be accurate, and using it is to be done entirely at your own risk. There have not been any reported mishaps in the entire history of OSDev.org, but in the odd chance you break something, we are not responsible.

To make the video card and monitor independent of each other, there is a standard in communication. This page describes the technical parts of that link with the necessary information to program a video card and have it render properly on the attached screen.

Contents

Display Signal

There are 15 pins in a standard VGA Cable. When your video card sends its video data to the monitor, it uses 5 data channels:

- Analog Red
- Analog Green
- Analog Blue
- Horizontal Sync
- Vertical Sync

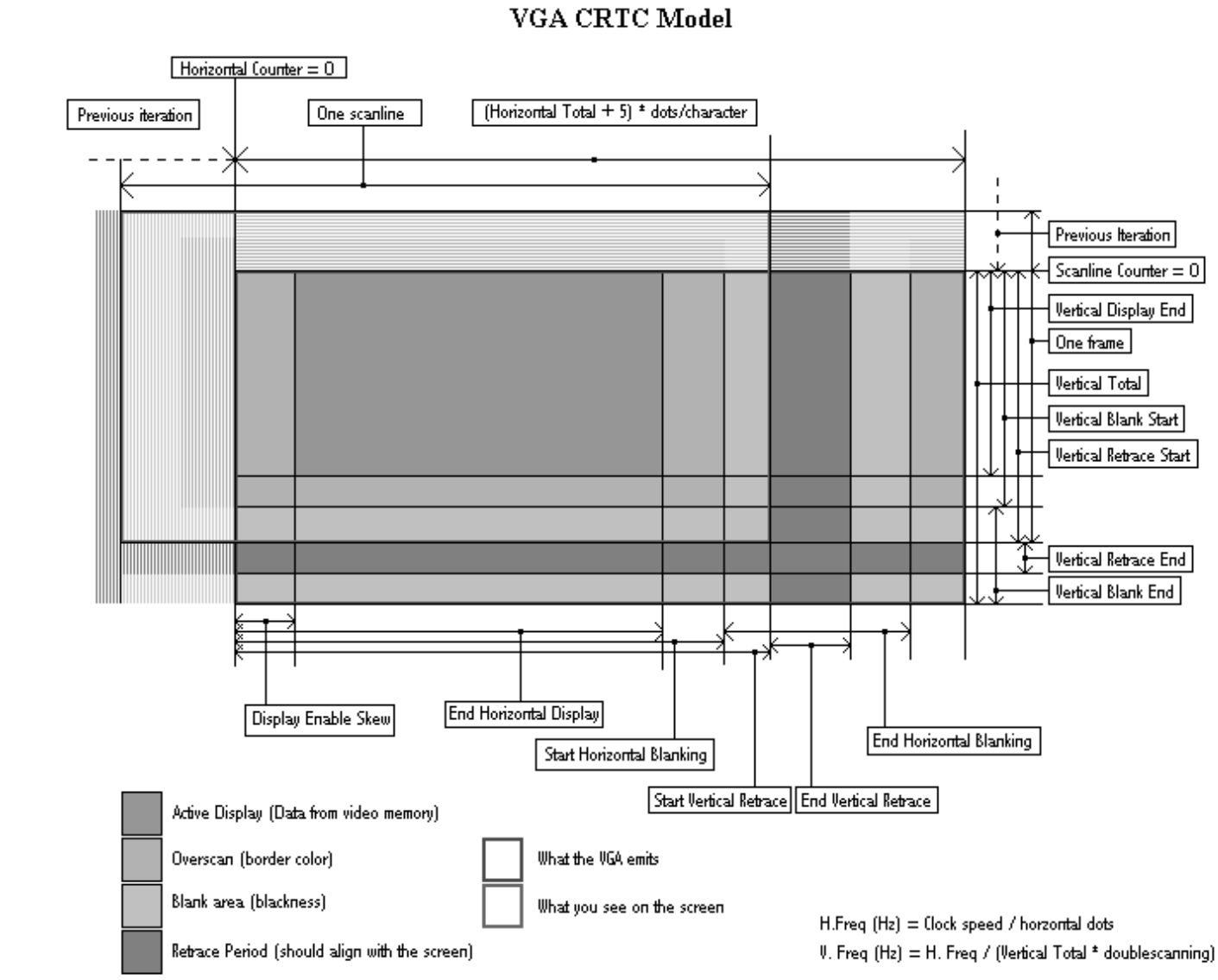
The screen is built up scanline by scanline, by sending the appropriate RGB values over the connection. However, only having a stream of colours will not tell you which part of the stream belongs to the top-left-pixel on the screen. To solve this they added two more signals: the horizontal sync which pulses when a horizontal line is done, and the vertical sync, which pulses when all rows have been completed. Thus, each frame corresponds to a single vertical synchronisation pulse. Since each frame has the same amount of pixels, the time between consecutive pulses is a constant. The monitor times the frequency of the pulses, then based on that it knows at what time the color data should be sent to what position on the screen.

The system is however not that simple. The first monitors were CRTs, which used magnetic fields to project electron beams onto a phosphoric layer, making them visible for a short period of time. The magnetic fields of a CRT had some inertia - they couldn't be set from a random location to another random location in the time available for one pixel. Hence the video signal has to have some gaps to cope with the time a monitor needs to alter the magnetic fields and point the electron beam back to the other side of the screen. In the meantime there could be no color signal, or you might have gotten stripes on the screen.

CRTs have improved a lot since then, and are now being superseded by the highly intelligent LCD display. The standard for signaling hasn't changed since. Since the original standard was made for CRTs, the rest of the document will implicitly assume an (old) CRT. LCD displays basically decode a CRT signal and then restructure it to fit their own screen grid. Similarly, in many other references you will see that the display unit of a video card is referred to as the "CRTC" or "CRT Controller"

Display Composition

All frames of a video signal have a specific layout, and video cards have a semi-standard way of thinking about these signals. Basically, you will have to provide the video card with enough information to be able to derive all sizes present in the following diagram. VGA Hardware explains how you can give these sizes to a VGA compatible. For other hardware, you should check your card's documentation on how these values are stored within them.



Both horizontally and vertically, there are 6 states, and hence, 6 sizes. Each size is in pixels. Most of the emitted data is part of the active display, the area of x resolution * y resolution pixels. Each of the cuts are where:

- active display goes into overscan
- where overscan goes into blanking
- where blanking gets the synchronization line changed
- where the sync is restored to its original value
- where the blanking goes back into overscan
- where the overscan goes into active display, where the scanline or frame is completed and the next one is started.

Video cards usually gives you the following registers to program (both horizontally and vertically)

- Resolution (pixel size of active display)
- Total (total number of 'pixels' in a single run)
- Blanking start and end (or start and size) - marks the location of the blanking. The are that is neither blanking nor active display becomes the overscan area.
- Sync start and end (or start and size) - marks the location of the synchronization pulse

Frequencies

When you increase the resolution, you will need to send more pixels to the display. If you would keep sending pixels at the same rate, the time to transmit one frame will go up, and consequently the amount of frames in a certain timespan will go down. Since a CRT displayed pixel only gives light for a short time before running out

of energy, it needs to be repeatedly refreshed. If this is done fast enough (at about 60Hz, 60 times a second) the screen appears almost constant to the human eye. This improves further when the refresh rate goes up to a point where it doesn't matter to the human eye. However when it drops too much, the screen starts appearing flashing, causing headaches to the user. Hence, we need to keep the frequency at at least 60Hz for user's sanity, and below some other rate dictated by the monitor's capabilities. To make a full frame of pixels fit within one sixtieth of a second, we will have to adjust the speed at which these pixels are transmitted. This speed is called the pixel clock, or dot clock. For example, a VGA's dot clock is either 25MHz or 28MHz, corresponding to 25 million pixels per second or 28 million pixels per second, the latter one being only just enough to display a resolution of 720x480 at 60Hz (recall that the active display is only a part of the frame). Most higher resolution video can therefore use a wide range of dot clocks, well above 25MHz, with the current range allowing enough bandwith to easily exceed 1600x1200 at a 100Hz.

While the resolution is limited by the video card, In most non-VGA scenario's, it is the monitor that can not handle the speed of the signal. The monitor has a allowed vertical frequency (the amount of frames per second, usually listed in Hz), and horizontal frequency (listed in KHz). Some CRTs are fixed frequency, only allowing certain frequencies to be used both horizontally and vertically. Old VGA displays are infamous for burning out when you feed them a signal that doesn't exactly match these rates. While modern CRTs are mostly protected from bad signaling, you must know that you can break hardware in this fashion, and that you need to be careful.

General Timing Formula

In order to cope with all the CRT antiquities, VESA has produced a set of equations that allows you to compute the various display settings you need, given the desired resolution. There are three separate sets of formulas: One to compute all settings from resolution and desired refresh rate, one taking resolution and horizontal frequency, and one taking resolution and dot clock.

Normally before setting a mode you would do the following:

- Compute the horizontal refresh rate and pixel clock from resolution and frequency
- Check the video card for the nearest pixel clock it can give you
- Compute all settings given the resolution and pixel clock.
- Check if the horizontal and vertical frequencies are within the monitor's allowed range.

If the frequencies happen to be out of range, you will have to adjust the refresh rate and or resolution. Since the refresh rate can go down a bit without giving issues, that's usually the first choice to fix. Hence:

- Clamp the horizontal frequency to its limit
- Compute pixel clock and vertical frequency from resolution and the new horizontal frequency
- Select the appropriate pixel clock the video card has (making sure you are rounding away from the limit, so round down if you use the maximum horizontal frequency)
- Compute the horizontal and vertical frequency given the new dot clock
- Check if the frequencies are within range.

The horizontal frequency should now be correct, you can decide whether or not to allow the new mode because the vertical rate has dropped below an acceptable rate, and you can inform the user that the mode is not supported.

The formula's provided by VESA use various scales of constants. Brendan's working on changing the scales to units. Right now, the scale space is still unknown.

Common parts of the GTF formulas

```
H_PIXELS_RND = ( ROUND ( H_PIXELS / CELL_GRAN_RND ) ) * CELL_GRAN_RND

if ( INTERLACE_REQUIRED == true )
{
```

```
      V_LINES_RND = ROUND ( V_LINES / 2 )
      V_FIELD_RATE_REQUIRED = REFRESH_RATE_REQUIRED * 2
      INTERLACE = 0.5

} else {

      V_LINES_RND = ROUND ( V_LINES) )
      V_FIELD_RATE_REQUIRED = REFRESH_RATE_REQUIRED
      INTERLACE = 0

}

if ( #MARGINS_REQUIRED == true ) {

      TOP_MARGIN_LINES = ROUND ( MARGIN_PERCENT / 100 * V_LINES_RND )
      BOTTOM_MARGIN_LINES = ROUND ( MARGIN_PERCENT / 100 * V_LINES_RND )
      LEFT_MARGIN_PIXELS = ( ROUND ( H_PIXELS_RND * MARGIN_PERCENT / 100 / CELL_GRAN_RND ) ) *
      CELL_GRAN_RND
      RIGHT_MARGIN_PIXELS = ( ROUND ( H_PIXELS_RND * MARGIN_PERCENT / 100 / CELL_GRAN_RND ) ) *
      CELL_GRAN_RND

} else {

      TOP_MARGIN_LINES = 0
      BOTTOM_MARGIN_LINES = 0
      LEFT_MARGIN_PIXELS = 0
      RIGHT_MARGIN_PIXELS = 0

}

/* use one of the GTF equations here */

V_FRONT_PORCH = MIN_PORCH_RND + INTERLACE
H_SYNC = ( ROUND ( H_SYNC_PERCENT / 100 * H_TOTAL / CELL_GRAN_RND ) ) * CELL_GRAN_RND
H_FRONT_PORCH = H_TOTAL / 2 - H_SYNC
H_BACK_PORCH = H_FRONT_PORCH + H_SYNC
```

GTF Using resolution and refresh rate

```
H_PERIOD_ESTIMATE = ( 1 / V_FIELD_RATE_REQUIRED - MIN_V_SYNC_AND_BACK_PORCH / 1000000 )

      / ( V_LINES_RND + 2 * TOP_MARGIN_LINES + MIN_PORCH_RND + INTERLACE ) * 1000000


V_SYNC_AND_BACK_PORCH = ROUND ( MIN_V_SYNC_AND_BACK_PORCH / H_PERIOD_ESTIMATE )

V_BACK_PORCH = V_SYNC_AND_BACK_PORCH - V_SYNC_RND

TOTAL_V_LINES = V_LINES_RND +

      TOP_MARGIN_LINES + BOTTOM_MARGIN_LINES +
      V_SYNC_AND_BACK_PORCH + INTERLACE + MIN_PORCH_RND


V_FIELD_RATE_ESTIMATE = 1000000 / H_PERIOD_ESTIMATE / TOTAL_V_LINES

H_PERIOD = H_PERIOD_ESTIMATE * V_FIELD_RATE_ESTIMATE / V_FIELD_RATE_REQUIRED

V_FIELD_RATE = 1000000 / H_PERIOD / TOTAL_V_LINES

if ( INTERLACE_REQUIRED == true ) {

      V_FRAME_RATE = V_FIELD_RATE / 2

} else {

      V_FRAME_RATE = V_FIELD_RATE

}
```

```
TOTAL_ACTIVE_PIXELS = H_PIXELS_RND + LEFT_MARGIN_PIXELS + RIGHT_MARGIN_PIXELS

IDEAL_DUTY_CYCLE = C_PRIME - M_PRIME * H_PERIOD / 1000

H_BLANK_PIXELS = ( ROUND (

            ( TOTAL_ACTIVE_PIXELS * IDEAL_DUTY_CYCLE / ( 100 - IDEAL_DUTY_CYCLE ) / ( 2 *
            CELL_GRAN_RND ) )

      ) ) * 2 * CELL_GRAN_RND

H_TOTAL = TOTAL_ACTIVE_PIXELS + H_BLANK_PIXELS

PIXEL_FREQ = H_TOTAL / H_PERIOD * 1000000

H_FREQ = 1 / H_PERIOD
```

GTF Using resolution and pixel clock

```
PIXEL_FREQ = PIXEL_FREQ_REQUIRED
TOTAL_ACTIVE_PIXELS = H_PIXELS_RND + RIGHT_MARGIN_PIXELS + LEFT_MARGIN_PIXELS

IDEAL_H_PERIOD = ( ( C_PRIME - 100 ) + ( SQRT ( ( ( 100 - C_PRIME ) ^ 2 ) +

      ( 0.4 * M_PRIME * ( TOTAL_ACTIVE_PIXELS + RIGHT_MARGIN_PIXELS + LEFT_MARGIN_PIXELS ) /
      PIXEL_FREQ / 1000000 ) ) )
      ) ) / 2 / M_PRIME * 1000

IDEAL_DUTY_CYCLE = C_PRIME - ( #M_PRIME * IDEAL_H_PERIOD / 1000 )

H_BLANK_PIXELS = ( ROUND ( TOTAL_ACTIVE_PIXELS * IDEAL_DUTY_CYCLE / ( 100 - IDEAL_DUTY_CYCLE ) / ( 2
* CELL_GRAN_RND ) ) ) * 2 * CELL_GRAN_RND

H_TOTAL = TOTAL_ACTIVE_PIXELS + H_BLANK_PIXELS
H_FREQ = PIXEL_FREQ / H_TOTAL
H_PERIOD = 1 / H_FREQ
V_SYNC_AND_BACK_PORCH = ROUND ( MIN_V_SYNC_AND_BACK_PORCH * H_FREQ / 1000000 )
V_BACK_PORCH = V_SYNC_AND_BACK_PORCH - V_SYNC_RND
TOTAL_V_LINES = V_LINES_RND + TOP_MARGIN_LINES + BOTTOM_MARGIN_LINES + INTERLACE +
V_SYNC_AND_BACK_PORCH + MIN_PORCH_RND
V_FIELD_RATE = H_FREQ / TOTAL_V_LINES

if ( INTERLACE_REQUIRED == true) {

      V_FRAME_RATE = V_FIELD_RATE / 2

} else {

      V_FRAME_RATE = V_FIELD_RATE

}
```

GTF Using resolution and horizontal frequency

```
H_FREQ = H_FREQ_REQUIRED
V_SYNC_AND_BACK_PORCH = ROUND ( MIN_V_SYNC_AND_BACK_PORCH * H_FREQ / 1000000 )
V_BACK_PORCH = V_SYNC_AND_BACK_PORCH - V_SYNC_RND
TOTAL_V_LINES = V_LINES_RND + TOP_MARGIN_LINES + BOTTOM_MARGIN_LINES + INTERLACE +
V_SYNC_AND_BACK_PORCH + MIN_PORCH_RND
V_FIELD_RATE = H_FREQ / TOTAL_V_LINES

if ( INTERLACE_REQUIRED == true)
{

      V_FRAME_RATE = V_FIELD_RATE / 2
```

```
} else {
      V_FRAME_RATE = V_FIELD_RATE
}
TOTAL_ACTIVE_PIXELS = H_PIXELS_RND + RIGHT_MARGIN_PIXELS + LEFT_MARGIN_PIXELS
IDEAL_DUTY_CYCLE = C_PRIME - ( M_PRIME / H_FREQ )
H_BLANK_PIXELS = ( ROUND ( TOTAL_ACTIVE_PIXELS * IDEAL_DUTY_CYCLE / ( 100 - IDEAL_DUTY_CYCLE ) / ( 2
* CELL_GRAN_RND ) ) ) * 2 * CELL_GRAN_RND
H_TOTAL = TOTAL_ACTIVE_PIXELS + H_BLANK_PIXELS
H_PERIOD = 1 / H_FREQ
PIXEL_FREQ = H_TOTAL * H_FREQ
```

Variable reference

These are the variables used in the equations above:

Hardware properties

These are constant for a given monitor. They describe the capabilities and limitations of a CRT.

C_PRIME

Defaults to 40, the actual value can be computed from EDID data.

M_PRIME

Defaults to 600, the actual value can be computed from EDID data.

CELL_GRAN_RND

Cell granularity - the amount of pixels the timings should be aligned to. For example, a VGA uses horizontal timings in multiples of the character clock, each character being 8 (graphics and some text modes) or 9 (most text modes) pixels wide. For a VGA graphics mode, you'll want to supply 8 here because of that. Depending on your specific hardware, the common values are either 8 or 1.

MIN_PORCH_RND

Usually 1. Probably used to force at least one unit of blanking around the sync period in order to not confuse graphics hardware that can not handle an absence of blanking around the synchronisation pulse.

MIN_V_SYNC_AND_BACK_PORCH

The time in microseconds the monitor needs to detect the vertical synchronisation signal and subsequently retrace to the top-left corner of the screen. Defaults to 550.

V_SYNC_RND

The amount of scanlines with the vertical synchronisation pulse active. **Todo** Seems to depend on something VESA doesn't tell you. Equals 3, although two scanlines are used in legacy VGA mode 12/13.

H_SYNC_PERCENT

The percentage of a scanline that should have the synchronisation pulse active. Equals 8%

= Video mode inputs

These form the request of the user/programmer.

MARGINS_REQUIRED

The margins are used to add a visible black surrounding to the displayed picture. I assume small margins are intended to make sure the entire picture is still visible when the picture isn't perfectly centered on the screen.

In theory it would be possible to use margins to simulate a small screen on a larger screen, or to simulate different aspect ratios perfectly. For example, if you were setting up a 640 * 480 video mode on a 16:10 display you could make the left and right margin 64 pixels each (with no top or bottom margins), so that it looks exactly the same as it would on a 4:3 display, rather than being stretched to fit. The opposite is also possible (e.g. displaying 16:10 video data on a 4:3 display with "letterboxing"). Unfortunately the GTF calculations don't actually support different sized left/right and top/bottom margins.

MARGIN_PERCENT

(Fixme: Percentage!) Amount of margins you want to have around your screen. VESA suggests a default of 1.8 to allow for screens that do not perfectly fit the image to the center.

INTERLACE_REQUIRED

There are two kinds of video modes, interlaced and progressive modes. Progressive modes display the full frame in one go, while interlaced modes display each other scanline in each run, taking two frames to render the complete picture. Normally, you don't want this, and several video cards don't support it, so you will normally want to supply **false** here.

V_LINES

Vertical resolution of the display you want. For a 640x480 resolution, supply 480.

H_PIXELS

Horizontal resolution of the display you want. For a 640x480 resolution, supply 640. Note that video cards will usually want this to be a multiple of eight because of cell granularity.

PIXEL_FREQ_REQUIRED

The speed of the dot clock (or pixel clock), in pixels per second. This is usually the same as the dot clock's frequency in Hz, but in some cases there's a factor two difference (2x the frequency when the clock is being doubled, half the frequency when pixels are emitted every other cycle) The dot clock is usually limited by the video card, as higher clocks are needed for higher framerates and resolutions.

REFRESH_RATE_REQUIRED

The amount of fields per second in Hz. This is also the number of frames per second for non-interlaced modes, or twice the number of frames per second for interlaced modes.

H_FREQ_REQUIRED

The desired horizontal frequency in Hz. In the most common use, you'll want this to be the maximum horizontal frequency of the monitor, to find the best refresh rate possible that doesn't violate monitor limitations.

Outputs

The final results of the GTF formula. These correspond to the parameters of the VGA CRTC model.

PIXEL_FREQ

The dot clock required (check units?)

H_TOTAL

Amount of pixels in a scanline

LEFT_MARGIN_PIXELS

Amount of margin at the left

RIGHT_MARGIN_PIXELS

Amount of margin at the right

H_TOTAL

The duration of horizontal sync in pixels

H_BACK_PORCH

The amount of pixels between the horizontal sync pulse and the start of the next scanline

H_BACK_PORCH

The amount of pixels between the end of active display and the start of the horizontal sync pulse

TOTAL_V_LINES

The amount of scanlines in a frame

TOP_MARGIN_LINES

Amount of margin at the top.

BOTTOM_MARGIN_LINES

Amount of margin at the bottom

V_BACK_PORCH

The amount of scanlines between the Vertical Sync pulse and the start of the next field

V_FRONT_PORCH

The amount of scanlines between the end of the field and the start of vertical retrace

The remaining three components needed to generate a complete signal are inputs. For reference, these are: H_PIXELS, V_LINES which form the resolution, and V_SYNC_RND, which is a constant for a given monitor.

Intermediates

Other results that are either byproducts, but may be useful for error and compatibility checking.

V_LINES_RND

Amount of active display scanlines per frame. Equals the Y resolution unless interlaced modes are used.

H_PIXELS_RND

Actual horizontal resolution used. Basically contains the X resolution rounded to the nearest allowed boundary.

V_FIELD_RATE_REQUIRED

Rate of complete images being sent to the monitor per second. Unless interlaced mode is used, equals the refresh rate.

H_FREQ

The horizontal frequency in Hz

H_PERIOD

The time in seconds spent on a scanline

H_PERIOD

The initial estimate of time in seconds spent on a scanline

INTERLACE

Interlaced modes are indicated by skipping half a scanline, so that the horizontal sync shifts half a period. Equals 0 or 0.5 depending on whether the interlace signal is included.

V_SYNC_AND_BACK_PORCH

The number of scanlines spent on VSync and bottom blanking

V_FIELD_RATE_ESTIMATE

The initial estimate of fields per second (check unit?)

V_FIELD_RATE

The number of fields per second (check unit?)

V_FRAME_RATE

The number of frames per second (check unit?)

TOTAL_ACTIVE_PIXELS

The amount of horizontally visible pixels, which equals the active display plus overscan

IDEAL_DUTY_CYCLE

Percentage. One of the steps to compute the amount of horizontal blanking.

IDEAL_H_PERIOD

Optimal time to spend on a scanline. This can usually not be achieved since a scanline is a integer number of pixels.

H_BLANK_PIXELS

The total amount of blank pixels horizontally (including horizontal sync)

Brendan's Sidenotes

Quoted:

I think part of the problem is that the original GTF formulas from VESA are copied from their spreadsheet, so that (for a silly example) instead of doing something like:

```
frequency = 10000000
period = 1 / frequency seconds
```

They'll do:

```
frequency_in_MHz = 10
period_in_ms = 1 / (frequency_in_MHz * 1000000) * 1000
```

And then simplify it to:

```
frequency_in_MHz = 10
period_in_ms = 1 / frequency_in_MHz / 1000
```

And then they'll remove any indication of what they've done:

```
frequency = 10
period = 1 / frequency / 1000
```

So that in the end, the formulas from VESA end up being a confusing mess where you're never too sure if there's a hidden scaling factor or not. Another simple example (that isn't made up) is "MARGINS_PERCENT", which is a value that would range from 0 to 50 that's typically divided by 100; and isn't a value that ranges from 0 to 0.5 that doesn't need scaling.

I plan on going through the formulas again, and making sure that all variables are either in pixels, lines, seconds or hertz (with no implied/hidden milliseconds, microseconds, kilohertz, megahertz or "percent" scaling factors).

I've also noticed that some of the calculations are the same in each formula. Rather than having 3 independent/larger pieces of code, it can be split into 3 smaller pieces of code that all use the same subroutines for common things. For example, you'd be able to replace about 10 lines from each formula with "call do_precalculations" to avoid code duplication (triplication?).

Also, I only posted "primary formulas". There's also a "secondary formula" where the results from the first set of calculations are used to find additional parameters; and there's another calculation (called the "blanking formula") for finding the values for "C_PRIME" and "M_PRIME" from some variables called C, M, J and K that's used for "secondary timings". I want to do the same for these formulas as I'm doing for the primary calculations. Then I need to go through all of these parameters and figure out which ones are actually useful (both for setting a video mode using VBE, and for setting a video mode in a "bare metal" video driver), and then work backwards removing redundant calculations (i.e. calculations that are only used to create information that you don't need anyway). After I've done this, I can combine it with the "common subroutines" idea, so that the entire "secondary formula" ends up being part of the "do_postcalculations()" routine, and so that the entire blanking formula becomes part of the "do_precalculations()" routine.

Then I want to find the valid ranges for all variables used in all calculations. This is mostly because of how I'm planning to implement it. For example, if I knew that "H_FREQ" ranges from 50000 to 1000000 then I'd be able to use "20:12" fixed point for this variable to improve the accuracy I get from a 32-bit integer.

Finally, all of these calculations can be simplified a lot if you assume that the "default GTF values" are being used. I want to provide simplified versions of the formulas, because most of the time you can use the default GTF values and can skip a lot of work.

See Also

Forum

- Getting a list of usable video modes - with emphasis on *usable*

External Links

- http://www.cs.unc.edu/Research/stc/FAQs/Video/GTF_V1R1.xls is one of the GTF spreadsheets you can find on the web.
- VGA connector (http://en.wikipedia.org/wiki/VGA_connector) on wikipedia
- EDID (http://en.wikipedia.org/wiki/EDID) on wikipedia