CHARACTERISATION OF THE HD6845 CRT CONTROLLER

UTTERCHAOS

1. Introduction

The HD6845 is commonly used as the CRT Controller (CRTC) in IBM CGA cards and whilst its documented behaviour is well-known, there are many questions remaining about how it operates.

Exploitation of "glitches" in the CRTC can lead to interesting effect used in demos such as 8088MPH and Area 5150.

Many of the internals of the 6845 can be guessed by examining the behaviour of the chip in a cycle exact way.

We can infer that registers Rx have a corresponding internal counter Cx which is compared with the value in Rx by some comparator. Actions may occur when the comparator goes high and when it goes low.

We know such internal counters must exist, otherwise the values in the registers would be lost during operation.

Of course the possibility remains that comparators compare with half or double values or that internal counters are incremented by 2, etc.

2. Notation

We use the terminology as standardised in the blog of Reenigne.

(1) hdot = time for one pixel in hires mode (640 horizontal pixels)
(2) ldot = 2 hdots = time for one pixel in 320 pixel mode
(3) cycle = 3 hdots = one processor cycle (aka ccycle)
(4) hchar = 8 hdots = one character in 80-column mode
(5) lchar = 16 hdots = one character in 40-column mode

3. Lockstep limitations

There are four things that can operate with a relative phase to one another: the CPU, the Programmable Interval Timer (PIT), the CGA card, the CRTC.

All are driven from the same clock 14.318MHz clock (each cycle is 1 hdot), but different dividers are used on the mainboard and on the CGA card.

Note that the CRTC derives its input from the CGA card, so these are by definition in sync. Also, the CGA card timing is relevant because of the wait states it asserts on the bus if the CPU tries to access VRAM. This is to avoid conflict with the CGA card accessing VRAM as it continually strobes through CGA memory for display and also DRAM refresh (which cannot be disabled, even in the blanking region).

At boot time the CGA card and PIT start with a random phase with respect to one another. As the PIT is run with a cycle of 12 hdots and the CRTC hchar is 8 hdots the best we can do is lockstep of the CRTC and PIT within 4 hdots, meaning there

is an unknown phase of 0-3 hdots between the two, which can only be changed on reboot.

This is relevant even if the PIT is not used as a system timer, as system DRAM refresh is run on timer 1. DRAM refresh can be slowed or sped up from its standard value of 18 PIT cycles (each scanline in standard modes 76 PIT cycles exactly), or temporarily turned off.

As the CPU clock is every 3 hdots and the 6845 hchar takes 8 hdots and these are relatively prime, it is possible to align these. In practice this is done by a series of delays and VRAM accesses and then waiting for a particular lchar in a CRTC frame. This puts the CRTC and CPU into lockstep (a consistent phase with respect to one another, though at any given time one doesn't know what this phase actually is).

A full CRT frame (262 scanlines of 57 lchars) is a precise number of CPU cycles and exactly 19912 PIT cycles. Of course most monitors have some tolerance and we are referring here to the CRT frame as set up in IBM BIOS modes.

Reenigne has developed code for lockstep to the maximum possible degree modulo the unalterable phase between the CGA card and CRTC.

Subject to the system timer not being used and DRAM refresh being off or only occuring when there is no other bus activity, this lockstep can be made cycle accurate.

We also make use of a partial lockstep in this document which consists of the second part of Reenigne's lockstep, followed by an additional section to bring things within one CPU cycle of lockstep, meaning that there is a consistent phase of 0-2 hdots which is not determined.

4. The Horizontal Displayed Register

This controls the number of characters that are displayed horizontally on each scanline, but many questions remain.

(1) Precisely when does it take effect?
(2) Is there any edge case behaviour?
(3) What resets the count to zero?
(4) What triggers when the comparator goes high/low?
(5) What happens if it is set multiple times in a scanline?
(6) What is its behaviour near other register values, such as horiz. sync.pos, and horiz. total.
(7) When is the horiz. disp. value added to the address?

4.1. **Methodology.**

The program HORDISP.ASM does the following to get a partial lockstep:

(1) Enter CGA graphics mode (equivalent of BIOS mode 4)
(2) Set up a CRTC frame with 2 scanlines of 2 lchars, with one lchar in the active region
(3) With a cycle exact loop of 144 cycles $= 4n - 3$ lchars ($n = 7$) wait until in the active region
(4) Waste some cycles until back in the inactive region
(5) With a cycle exact loop of $4k$ lchars minus 1 cycle, wait until in the active region again

This puts the CPU in a known state with respect to the CRTC, within 1 cycle (3 hdots) every time.

The program turns off the keyboard and timer interrupts (the only ones firing on a quiet XT) and polls the IRR to see if a keyboard IRQ has occurred. If this is detected, the entire program restarts, usually modifying some program values before doing so, in response to the keypress.

The main loop can be adjusted to the precise number of cycles for a standard CGA CRT frame and DRAM refresh is set to 38 PIT cycles to localise interference to the same two positions on each scanline.

The keyboard polling is also done at a consistent time about half way down the frame and its position can be adjusted in a cycle accurate way.

Longrunning instructions such as SHL (with a high count) and MUL are used for timing within 4 and 1 cycles respectively. These instructions have no bus access associated with them, except for instruction prefetch. The code is written so that as many instruction prefetches as practicable occur at the same relative place on a scanline to minimise interference from system DRAM refresh.

The program allows setting the horiz. disp. reg. at any time to within about 1 cycle. Often simply restarting by pressing a key causes a different phase (0-2 hdots) at the point the register is being written, due to the imperfect lockstep, allowing easy exploration of all three behaviours.

The program has two modes, one which sets the background register of the CGA to put a short piece of red background on a scanline at two places on the screen. The first pixel after this red mark is taken to be the point at which this register write took place.

This can then be swapped for code that writes a CRTC register instead (the horiz. disp. reg.) at precisely the same point (by using the same code just with different values for the register port address and value).

This allows one to visually control almost down to the pixel (really to within 1 CPU cycle) where the CRTC register write happens.

Once approximate information is gained in this way, we may write a program that uses full lockstep and precise timing to get information down to the hdot.

4.2. **When does the horiz. disp. reg. value take effect?** We do not know precisely when the background colour change occurs relative to when the register is written, and there is a similar ambiguity for the CRTC register write. Therefore all we can really report without instrumentation is when the CRTC register write *appears* to take place as determined by the visual change that a similar background register write causes when done at precisely the same time in the loop (modulo the cycle uncertainty - the program restarts when switching from a background register write to a CRTC register write).

We see three behaviours which can be described as follows:

(1) The horiz. disp. value (n) takes full effect, blanking all but the first n characters
(2) The horiz. disp. value does not take effect, causing all characters to be displayed right out to horiz. total +1 characters, with the exception of the final character which only displays the first hchar of the full lchar, the right hand hchar being blank
(3) An intermediate behaviour where the horiz. disp. value only takes effect from the second half of the lchar where it is supposed to start blanking, leaving an extra hchar displayed

As far as can be determined using HORDISP.ASM, the dividing line between the three behaviours occurs when the register write occurs on the second last pixel of character $n-1$ (numbered from 0) where n is the value being written into the horiz. disp. reg., i.e. the second last pixel before it is supposed to take effect.

The precise hdot and for how many hdots (1 or 2) that the intermediate behaviour occurs is only known approximately.

Given that all three behaviours can be exibited within the same cycle it is likely that there is only one hdot exhibiting the intermediate behaviour and experiments suggest this could be the third last hdot of the lchar. This however remains to be confirmed.

4.3. **Is there any edge case behaviour?** If horiz. disp. is set greater than or equal to horiz. total +1 the final character of the scanline is only half displayed. The second hchar is blank.

This is probably a general behaviour of this CRTC (the final hchar before the end of the scanline is blanked).

4.4. **What resets the count to zero?** Unknown.

4.5. **What triggers when the comparator goes high/low?** Unknown.

4.6. **What happens if it is set multiple times in a scanline?** Unknown.

4.7. **What is its behaviour near other register values, such as horiz. sync.pos, and horiz. total.** Unknown.

4.8. **When is the horiz. disp. value added to the address?** Unknown.