TMS 99xx TIMING

The way the VDP works is a sort of a "best effort" approach. When you write data to the data port, the VDP will copy that data to VRAM when it's ready to, and not before. If the CPU writes a new piece of data before the VDP does the move, there is no error, interrupt, or status bit to check, the old data is replaced and when the VDP comes to get it, the new data is moved instead of the old data.

But in the VBL when the TMS has nothing to display, you can write as fast as possible.

After an address is written, the VDP needs 2uS to retrieve the data. Between reads, the VDP needs 8uS to get the next piece of data.

And now, the variable part - the VDP only provides a CPU access window from every 0 to 6uS.

So the data says that it's fine to write the address and even read the first piece of data immediately on our hardware, because we will take longer than 2uS anyway.

In the vertical blank, and if the blanking bit has disabled the screen, there is no wait for a CPU access window - so you can access VRAM at top speed. Between reads you need to wait 8uS to guarantee success. Fast code running in zero wait-state memory (scratchpad or modified) may be able to send data faster than every 8uS (indeed this was what I achieved, managing a write roughly every 5uS). This will fail.

If you look to the AppendexE from the TMS99xx pdf and read the examples for the TMS99xx cpu you see that here are also NOP`s and the comment is "dont write to fast" because this CPU is also very fast. On Z80 machines this is not needed because the RAM access itself is so slow that the TMS is always ready.

But at then end best write only in the VBI.

Some more information from an TI thread in AtariAge, maybe interesting. Here the same, only fast machines can overrun the VDP. Only access in VBI i save. But this all only if you want access to the VDP outside the VBI.

When the VBI start the VDP give you 4300 microsecond for access at full speed.

The VDP delay question, we had a huge investigation on the Yahoo group a few years ago. I'm the one who put a logic analyzer on the bus and measured instruction and turnaround time to the VDP. On a standard console, whether you are in scratchpad RAM or not, writes can not overrun the VDP because of the additional delay induced by the CPU's read-before-write cycle. Reads appear to be unable to overrun the VDP, unless you use the fastest possible instruction time (IIRC, it was MOVB R0,R1 -- note no indirection and no increment). However, there is a case where after setting the VDP address, and then doing a fast read instruction (such as one involving only registers), you MIGHT overrun the VDP. (For instance, write the address then immediately do a MOVB R1,<anything>, where R1 points to the VDP read data address). This is because the time between the write to the VDP for setting the address and the read from the VDP for getting the byte may be too short. Using an absolute address (MOVB @VDPRD,<anything>) seems to give you enough time with the extra memory read to be safe. As Mizapf notes, faster machines like the Geneve may need a delay (note the Geneve also has a different video chip), and accelerated TI consoles (faster crystal) may as well. These are relatively rare, however.

Emulators do not appear to care about the delay today. Classic99 does not at this time.

There ARE periods at which the delay is not needed. After VSYNC is the only predictable one (because there is no external indication of the other windows). The TMS9918 data manual notes that there are 4300uS of CPU access after a vertical blank starts -- some of this time is eaten by the TI interrupt routine if you leave it on. If the blank bit is enabled, so that the display is not being drawn, the CPU also gets full speed access to memory.

The reason for this is that CPU accesses to VRAM are given 'access windows' depending on the VDP's exact mode. There is always a 2uS delay, and then additional time is added depending on the current mode. You'll find the table on page 2-4 of the datasheet, but basically, in text mode you need 3.1uS, in bitmap or graphics mode you need 8uS, in multicolor you need 3.5uS, and when the display is off or during vertical blank, you need 2uS. Most 9900 instructions take far longer than these times, and so TI's recommendation was pretty much just a guarantee, as well as future-proofing for future, faster CPUs.

2. RMW - as Mizapf notes, there's no way to do it. If possible, it may be helpful to 'double buffer', keep a copy of a VDP memory area in CPU RAM, and then you only need to push, never read. I'm not sure that helps you for a line draw function, though.

3. Mentioned a few times, but yeah, SOC, SZC, ANDI and ORI are your best bet for bit manipulation. You can use a table of bits with SOC or SZC to remove the need for shifting

4. LI vs MOV is well covered. As an extra benefit, LI does not do a read-before-write when loading the register, one of the rare instructions that doesn't. When MOV can be considered for performance is when you replace @ONE with a register. Because memory performance is the biggest bottleneck on the TI, often the /length/ of your instruction is more important than the work it does.

5. Byte instructions on the 9900 are the same speed except when doing Workspace Register indirect autoincrement (*R1+) - a byte operation is 2 cycles faster.