# Vector Graphic 8 inch Floppy Controller

The floppy controller does not support DMA, so the CPU must be able to keep up with the double density transfer rate. A data byte is transferred every 16us, but due to internal timing of the 1793 controller IC and board timing, worst case read timing is 13us and 11us for writes.

Like many controllers that support programmed data transfer, this controller provides a WAIT register that stalls the processor until the next byte from the floppy is available/needed. However, the WAIT register cannot be used until *after* the 1st byte is received/requested. The processor must poll the DRQ (data request) status register and load the first byte once DRQ is asserted. All remaining bytes of the transfer can be then timed using the WAIT register.

Vector Graphic computers use a Z80 processor and DRAM boards that utilize the Z80 for refresh. Refresh occurs during the instruction fetch cycle, so the Z80 cannot be suspended for long or DRAM refresh will fail. For this reason, the floppy controller uses a one-shot to limit the WAIT time. Once data transfer begins, this is plenty of time, however, the initial byte of a read or write request may take a full disk revolution (167ms) or more, so WAIT cannot be used for the 1st byte.

On the Rev 0 board a 2200pf cap is used with the one-shot delay of about 40us. This is short enough that the one-shot times out during single density reads if waiting for byte 129 (instead of using a counter and exiting the read loop). This is because the 1793 reads two more CRC bytes after the last data byte (64us) before issuing the command completion IRQ. The Rev 1 board uses a .022uf cap for a 250us+ delay. The Rev 1 schematic shows a 1000pf capacitor for the one shot, but I have not seen a Rev board 1 with 1000pf or a technical advisory that changed from 1000pf to .022uf. It’s probably a typo on the schematic.

Meeting worst-case timing while polling DRQ for the 1st data transfer is difficult. In order to meet timing requirements, the DRQ status register and software were designed together to make a very short polling loop. This loop is shown below. The code must be at offset FCh-FFh within a page. Prior to the loop, H is loaded with the page address and C is loaded with the board status register port address (for the IN L instruction). Reading the DRQ status register returns FCh when DRQ is false and FFh when DRQ is true (the DRQ line is connected to bits 0 and 1).

```
xxFC     IN     L            ;load board status register into L
xxFE     JP     (HL)         ;jump to xxFC (DRQ false) or xxFF (DRQ true)

xxFF     IN     FDCDATA      ;read the first data byte
```

Assertion of DRQ is asynchronous to a CPU bus cycle. For this reason, board revision 1 latches DRQ with a flip-flop clocked by bus clock phi 2. However, board revision 0 did not include this flip-flop. Therefore, DRQ could be in flux at the instant the CPU reads the DRQ status register. The resulting read could have values of FDh or FEh in addition to the expected values of FCh and FFh. If FDh is returned, the jump (HL) instruction jumps to the 2nd byte of the IN L instruction. If FEh is returned, the jump (HL) instruction jumps to itself and hangs in an infinite loop.

To get around this race condition, the revision 0 board includes a blue-wire mod that connects the DRQ line to status register bit 1 through an AND gate used as a buffer to delay the DRQ signal by one TTL delay. This results in only one value, FDh, being returned during the race condition instead of both FDh or FEh. This causes the jump (HL) instruction to jump to the 2nd byte of the IN L instruction. The second byte of this instruction happens to be a LD L,(B) instruction. By having B pre-loaded with FFh, the code properly jumps to xxFF during the race condition.