Microdrives – The Inner Workings

Tobias Fröschle, QL Forum

Before anything can happen, a microdrive needs to be formatted - The QL basically writes headers, followed by data blocks, with gaps in-between.

A microdrive sector header is 26 bytes long and mainly contains the sector number of the following data sector and the medium name.

The formatting routine writes headers and sectors (in descending order from 255 down) onto the tape multiple times, then tries to find the highest sector number that actually got onto the tape (as the sectors are written in backwards order, and a cartridge can't hold much more than around 230 sectors, the higher sector numbers become overwritten at the end of the formatting run because of the tape actually being a loop). The highest sector number found is what QDOS prints after FORMAT mdv_x as overall # of sectors. Note that the sector headers will never be written to again after the formatting run (That is why there is gaps between them and the data - To make sure we don't accidentally hit a header instead of data).

What we have now, is a loop of tape filled with 26-byte headers and 512-byte (empty, but formatted) sectors in-between. Sectors and data blocks are interspersed with empty lengths of tape - The so-called gap. The gap is important, because the microdrive circuitry contains mechanisms to distinguish random noise from the read head (the gaps) from actual data blocks (headers or sectors) on the tape - This allows the ZX8302 to fire an interrupt to the 68008 when a gap runs by on the tape.

Registers
$18002 - Transmit register. Bit 4 controls whether we talk to network or microdrive. 1 means microdrive. Note this register is shared between the 8302 and the IPC. Also note those bits cannot be read back from the register and also other bits are used to talk to the IPC. That is why the QL always has a valid copy of what this register should be in the system variable $a0. Write $10 to that register to talk to the microdrive (after having made sure no IPC comms takes place at the moment by waiting some milliseconds in supervisor mode)

$18020 - microdrive control register (write)
bit 0 - select bit - a kind of token that can be shifted through the chain of drives selected together with
bit 1 - the select clock bit
(You set the select bit to the desired state and toggle the clock bit <drive number> times. This works like a shift register across the drives) The drive that has the token after that is the drive for future communications)
Once a drive is selected, the motor will also run.
bit 2 - read or write mdv (write is 1)
bit 3 - erase tape (erase is 1)

$18020 - microdrive status register (same as above, but read)
bit 1 - microdrive transmit buffer full (you are only allowed to write the next byte to the drive if this is 0)
bit 2 - microdrive receive buffer ready (there's a byte to pick up)
bit 3 - tape gap present

$18022 - transmit data (write a byte here to put it on the tape)
$18022 - microdrive read track 1
$18023 - microdrive read track 2

$18021 - interrupt control/status (bits 0 and 5 enable/disable the gap interrupt)

Now a short description of how this is all working together:

Assume you want to read a sector, say "21" from the microdrive 2.

1. Start up the drive by clocking 1 "1" into drive 2 (set select bit, then toggle clock bit 2 times)
2. wait some time for the drive to spin up, enable read
3. wait for a gap interrupt
4. start reading from $18022/$18033 - if it's a header, check for sector #21
5. goto (4) if it's not the correct sector header (basically wait for sector header #21 to come by)
6. wait one more gap (data sector comes by)
7. read 513 bytes (512 bytes data + 1 byte checksum)
8. spin down the drive by clocking in a "0" into drive 2.

Writing works the other way round (make sure you never try to write to a sector header - This will corrupt the tape), with some involvement of the erase head, I think, to protect the gaps. Also make sure you enable write only after you have seen a gap, a header, and another gap.

Obviously, there's also a bit of timing involved here - The QL ROMs do this with fine-tuned tight loops. Note this will not work in RAM with unexpanded QLs in contended memory because of the 8301 getting in the way for video output. Only from ROM or upper memory.

Also, I have disregarded the fact that the drive actually works with two tracks. But that doesn't change much in principle.