XD Media Specification

Looks like I'm the first poor sod to implement XD in software. It seems to be a closed-spec, can't even find a forum or design group for it (it was designed by Fuji and Olympus).

Fortunately, XD is very similar to SmartMedia. The details listed in this document are based on my own reverse engineering, and Linux's SDDR09 driver (a software implementation of SmartMedia reader). You can find more details in the SmartMedia "sample" specifications∞ but information there can't be republished.

As this is somewhat reverse engineered and somewhat guesswork, there may be inaccuracies/incompleteness here!

Media Signature

The media stores a 4-byte signature. This is identical to recent SmartMedia signatures.

Byte 0: Manufacturer ID

This seems to be the same ID type as SmartMedia.

Byte 1: Device ID

This ID represents the card capacity, ala SmartMedia.

Byte 2: 128-bit ID availability

This byte will read 0xA5 when a 128-bit media ID is available.

Byte 3: Extra read media ID command availbility

This byte will read 0xC0 when another read-media-signature command is available.

Address Space

NOTE: Unlike SmartMedia, I think that sector-size/block-size is constant accross all media, but this is only a guess based on the fact that my 16mb card behaves the same as my 256mb card!

- Data is separated into pages (sectors). One page is 512 bytes.
- Each page has 16 bytes of control information associated with it.
- Pages are grouped into blocks. One block is 32 pages.
- Blocks are groups into zones. One zone contains 1024 blocks.

- Only 1000 blocks can be used per-zone.
- This leaves you with 24 'spares' which can be used when a block dies

- I'm not exactly sure how we should cope with a zone which has less than 1000 usable blocks..?

- Data starts at physical block (PBA) 2
- Sector 0 at PBA 1 reads the 128 byte CIS (Card Information Structure) of the card, followed by 128 junk bytes, followed by the last 256 bytes repeated again.

- In hardware, memory is implemented in physical blocks (PBAs)
- PBA's are not used sequentially, each used PBA contains a logical (LBA) address.
- For example, logical block 0 may be stored at physical block 813
- Not all PBA's will hold a LBA address, as PBA's can be unused or reserved.
- A software driver needs to keep pba-to-lba and lba-to-pba lookup tables

- The LBA address stored per-PBA is always based on 1000 offsets.
- For example, if a PBA in zone 1 reads an LBA value of 312, then that block is actually LBA 1312
- Another example, if a PBA in zone 8 reads LBA value 641, that block is actually LBA 8641
- The above allows you to be clever when figuring out which zone a LBA will exist in, e.g. LBA 9645 will exist in zone 9. This will help when building lba-pba mapping tables on-demand.

- If a block reads a raw LBA value greater than 1000, you treat it as a spare block.

Page Control Data

- Every page has 16 bytes of control data associated with it.
- The control data indicates the usability of the parent PBA (unused, damaged, reserved, etc...)
- The control data also stores two checksums for the data stored in the page
- Each page data is split up into two 256 byte 'chunks', and a checksum is assigned to each

- The control data stores a 10-bit LBA index for the PBA
- Obviously all sectors in the same PBA will give the same LBA index and PBA usability info

- The Address Field bytes (6,7) are repeated in the control data (as 11,12), both instances should be identical
- The Address Field bytes should have even parity when strung together (`byte(6) ^ byte(7) == 0`)

Bytes 0,1,2,3: Reserved(?)

In a 'normal' data PBA, the control data starts with four 0xFF values. If this is not the case, then assume that the PBA is reserved and can't be used for storing data.

Byte 4: Data Status

In a 'normal' data PBA, this byte should read 0xFF. If it doesn't, assume the PBA is dead.

Byte 5: Block Status

In a 'normal' data PBA, this byte should read 0xFF. If it doesn't, assume the PBA is dead.

Byte 6: Address Field 1

The second word in this byte must read be 0x01. If it isn't, assume the PBA is dead.
i.e.: `bit(7)=0, bit(6)=0, bit(5)=0, bit(4)=1`

Bit 3 should be set to 0.
The remaining bits (2,1,0) represent the upper 3 bits (9,8,7) of the LBA address of the block.

Byte 7: Address Field 2

Bits 7 to 1 represent the lower 7 bits (6-0 respectively) of the LBA address of the block. Bit 0 is a parity bit.

Bytes 8,9,10: ECC Checksum for Chunk 2

3-byte checksum for the second chunk of page data.

Byte 11: Address Field 1 Copy

Always the same as Address Field 1.

Byte 12: Address Field 2 Copy

Always the same as Address Field 2.

Bytes 13,14,15: ECC Checksum for Chunk 1

3-byte checksum for the first chunk of page data.

ECC Checksum

TODO: Describe me
