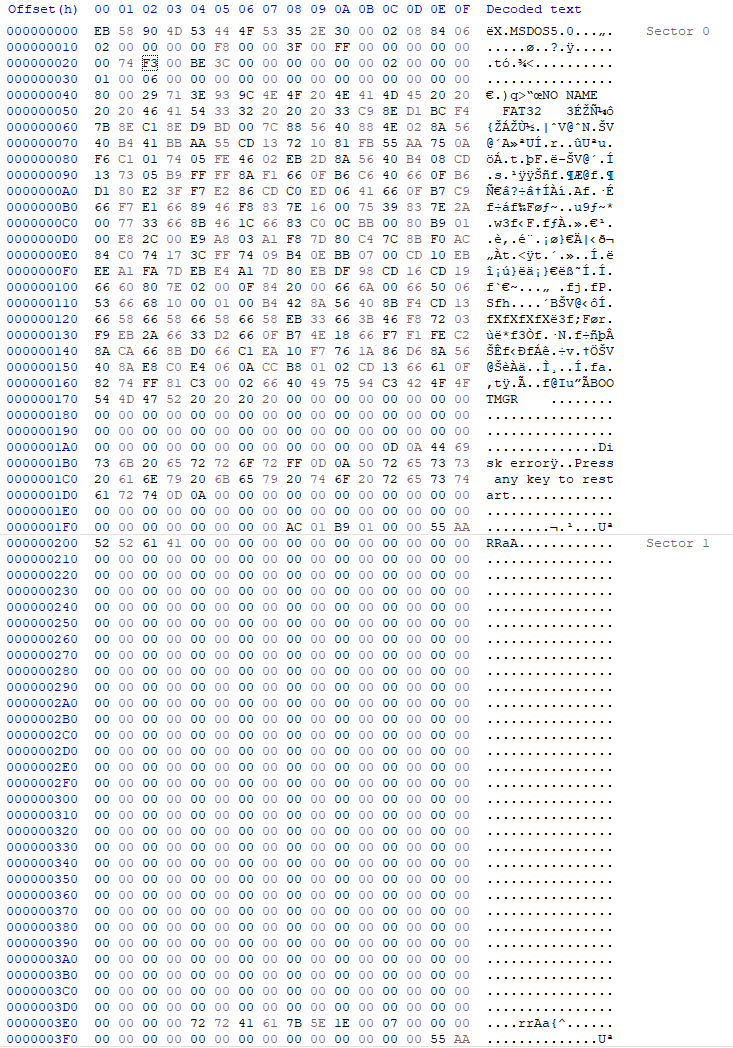
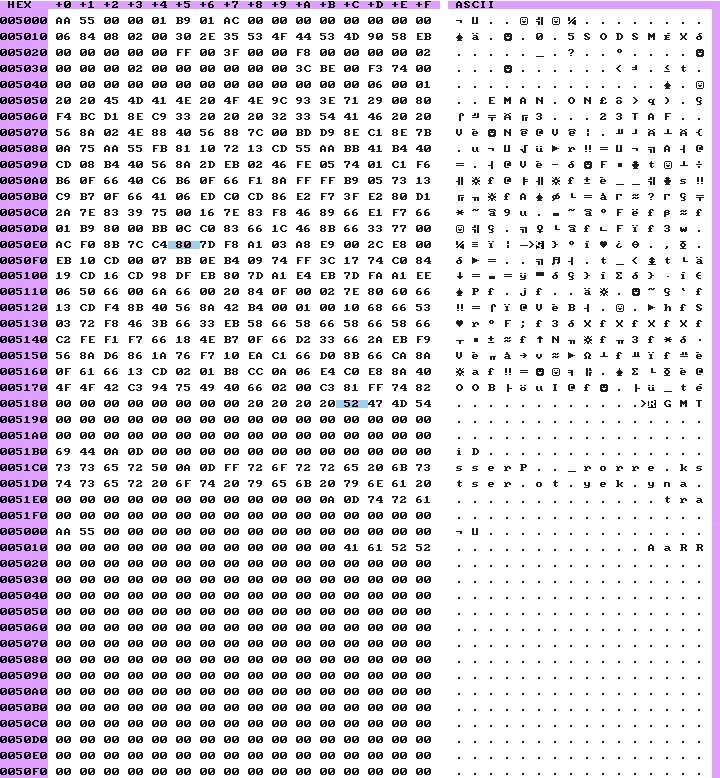
# SDInterface Debugging

## Reader Byte Order

SD reads are returning odd byte-order. Here’s what HxD shows should be the contents of Sectors 0 and 1:



And here’s what’s being read to the SD buffer in DDR3 RAM:

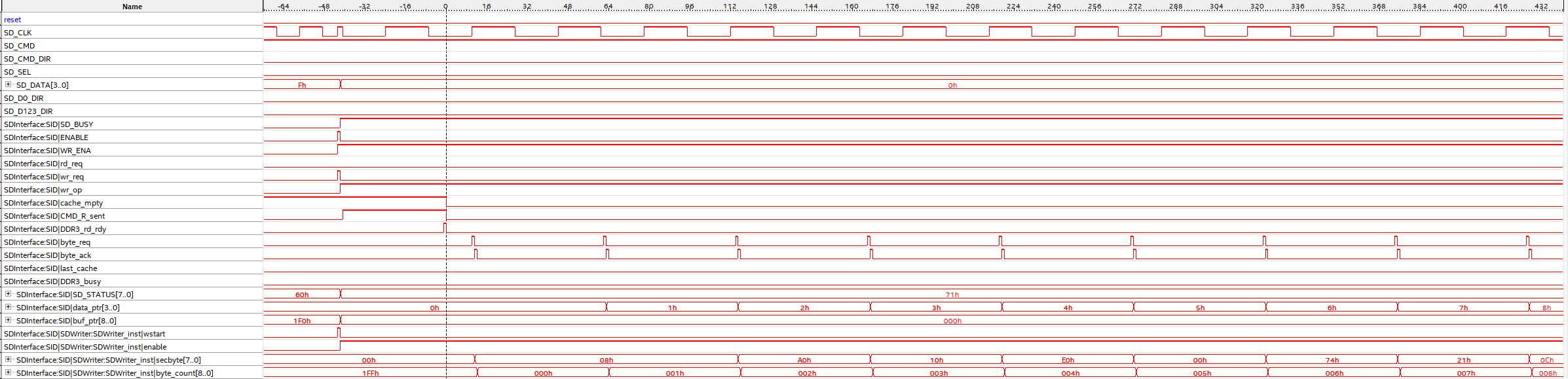


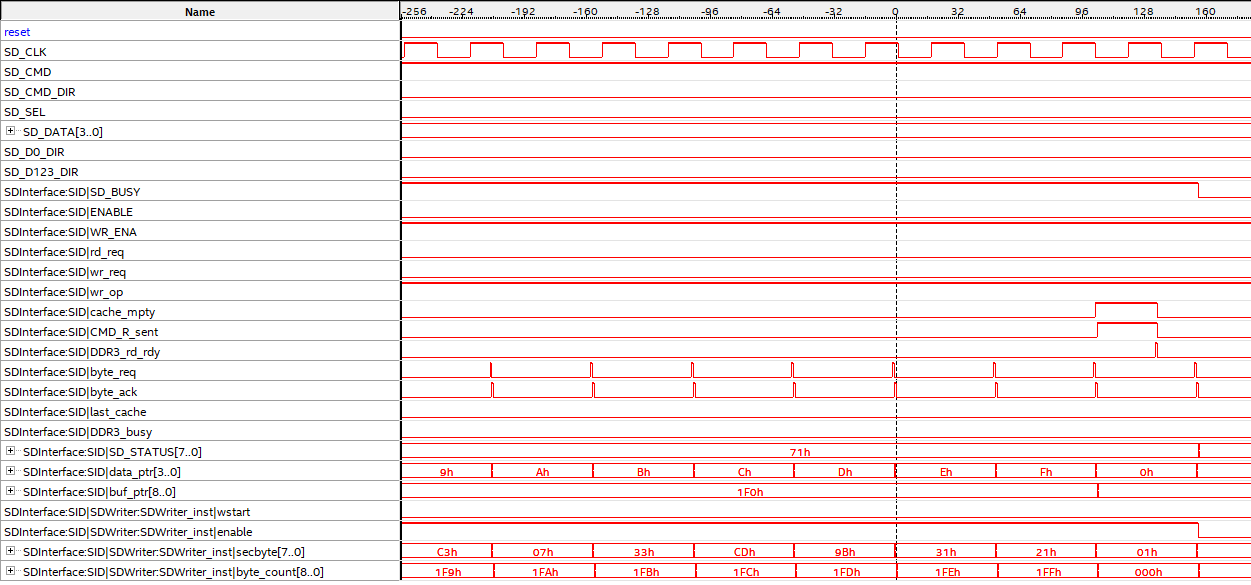
**Two issues are immediately obvious –** the last 16 bytes are being written to RAM first, and every 16-byte sequence is reversed. The bytes themselves are fine, I just need to reverse the order that the bytes are written to the cache in SDReader.sv.

The second issue is solved by setting **buf\_ptr**’s starting/reset value to **0xFF0**. It starts incrementing with the first byte received, so by the time the cache is filled for the first time, the first 16 bytes of data are written to address *buffer + 0x10*. By starting **buf\_ptr** at **0xFF0**, the first read will write the cache to *buffer + 0* (because 9-bit **buf\_ptr** rolls over from **0xFFF** to **0x000**), where it should start.

## SDInterface – WriterByteStream Development

The next issue is fixing alignment errors in the Writer ByteStream – the stream of bytes sent from the SDInterface module to the SDWriter module. As can be seen below, data\_ptr, byte\_count and secbyte aren’t tracking each other properly. Secbyte should be 0x00 when byte\_count is 0x00 and 0x08 when byte\_count is 0x01.



The end of the write transaction looks like this:

Still missing last byte at end of 512 block. This was partly to do with the location where buf\_ptr was being incremented and some legacy tweaking code that I needed to remove (first\_byte wasn’t necessary).

The last problem to do with the Writer ByteStream is that it is non-repeatable. Once an SD WRITE operation is attempted, no further ops will execute until the system is reset. At first look, it appears to be an issue with these lines in SDInterface (84 & 85):

assign rd\_req       = ( !BUSY && !WR\_ENA ) ? ENABLE : 1'b0 ; // only pass rd\_req if interface isn't busy

assign wr\_req       = ( !BUSY && WR\_ENA  ) ? ENABLE : 1'b0 ; // only pass wr\_req if interface isn't busy

ENABLE is going HIGH, WR\_ENA is either HIGH or LOW depending on the value written to port 242 for a WRITE or READ accordingly. BUSY is also LOW at the same time – therefore, rd\_req and wr\_req should have a valid value, but they do not.

Turns out it was the BUSY signal in SDInterface not being reset by the write op. Simples.

## Read Data Alignment Problem

This issue has arisen again after switching back from the test SDInterface module to the older (read-only) one.

D000 58 90 4D 53 44 4F 53 35 2E 30 00 02 08 84 06 02 X.MSDOS5.0......

D010 00 00 00 00 F8 00 00 3F 00 FF 00 00 00 00 00 00 .......?........

D020 74 F3 00 BE 3C 00 00 00 00 00 00 02 00 00 00 01 t...<...........

D030 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 80 ................

D040 00 29 71 3E 93 9C 4E 4F 20 4E 41 4D 45 20 20 20 .)q>..NO NAME

D050 20 46 41 54 33 32 20 20 20 33 C9 8E D1 BC F4 7B FAT32 3.....{

D060 8E C1 8E D9 BD 00 7C 88 56 40 88 4E 02 8A 56 40 ......|.V@.N..V@

D070 B4 41 BB AA 55 CD 13 72 10 81 FB 55 AA 75 0A F6 .A..U..r...U.u..

D080 C1 01 74 05 FE 46 02 EB 2D 8A 56 40 B4 08 CD 13 ..t..F..-.V@....

D090 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66 0F B6 D1 s......f...@f...

D0A0 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F B7 C9 66 ..?.......Af...f

D0B0 F7 E1 66 89 46 F8 83 7E 16 00 75 39 83 7E 2A 00 ..f.F..~..u9.~\*.

D0C0 77 33 66 8B 46 1C 66 83 C0 0C BB 00 80 B9 01 00 w3f.F.f.........

D0D0 E8 2C 00 E9 A8 03 A1 F8 7D 80 C4 7C 8B F0 AC 84 .,......}..|....

D0E0 C0 74 17 3C FF 74 09 B4 0E BB 07 00 CD 10 EB EE .t.<.t..........

D0F0 A1 FA 7D EB E4 A1 7D 80 EB DF 98 CD 16 CD 19 66 ..}...}........f

D100 60 80 7E 02 00 0F 84 20 00 66 6A 00 66 50 06 53 `.~.... .fj.fP.S

D110 66 68 10 00 01 00 B4 42 8A 56 40 8B F4 CD 13 66 fh.....B.V@....f

D120 58 66 58 66 58 66 58 EB 33 66 3B 46 F8 72 03 F9 XfXfXfX.3f;F.r..

D130 EB 2A 66 33 D2 66 0F B7 4E 18 66 F7 F1 FE C2 8A .\*f3.f..N.f.....

D140 CA 66 8B D0 66 C1 EA 10 F7 76 1A 86 D6 8A 56 40 .f..f....v....V@

D150 8A E8 C0 E4 06 0A CC B8 01 02 CD 13 66 61 0F 82 ............fa..

D160 74 FF 81 C3 00 02 66 40 49 75 94 C3 42 4F 4F 54 t.....f@Iu..BOOT

D170 4D 47 52 20 20 20 20 00 00 00 00 00 00 00 00 00 MGR .........

D180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

D190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

D1A0 00 00 00 00 00 00 00 00 00 00 00 0D 0A 44 69 73 .............Dis

D1B0 6B 20 65 72 72 6F 72 FF 0D 0A 50 72 65 73 73 20 k error...Press

D1C0 61 6E 79 20 6B 65 79 20 74 6F 20 72 65 73 74 61 any key to resta

D1D0 72 74 0D 0A 00 00 00 00 00 00 00 00 00 00 00 00 rt..............

D1E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

D1F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................

The data\_cache write order appears to be out – 00, 0F, 0E…

Turns out data\_ptr was not being set to the correct initial value due to the use of **1’hF** as the value, which set it to 1 instead of F. Instead, using **‘hF** fixed the issue and set it to F as intended.

## SD MBR Contents

|  |  |  |
| --- | --- | --- |
| Offset (hex) | Size | Content |
| 1FE | 2 | If 0x55 AA – Boot Sector |
| 0 | 1 | If EB or E9 – Is DBR |
| 1C6 | 4 | DBR Sector No. (reverse order / LSB first) |
| B | 2 | Bytes per Sector (LSB first) |
| D | 1 | Sectors per Cluster |
| E | 2 | Reserved Sectors (LSB first) |
| 10 | 1 | FAT version |
| 11 | 2 | Root dir item count (LSB first) |
| 16 | 2 | Sectors per FAT (LSB first)  > 0 – **FAT16**  = 0 and offset 56 = 32 – **FAT32**  Otherwise – **Unknown Filesystem**  If FAT32:  Sectors\_per\_FAT = 27, 26, 25, 24  Root\_Cluster = 2F, 2E, 2D, 2C |
|  |  |  |