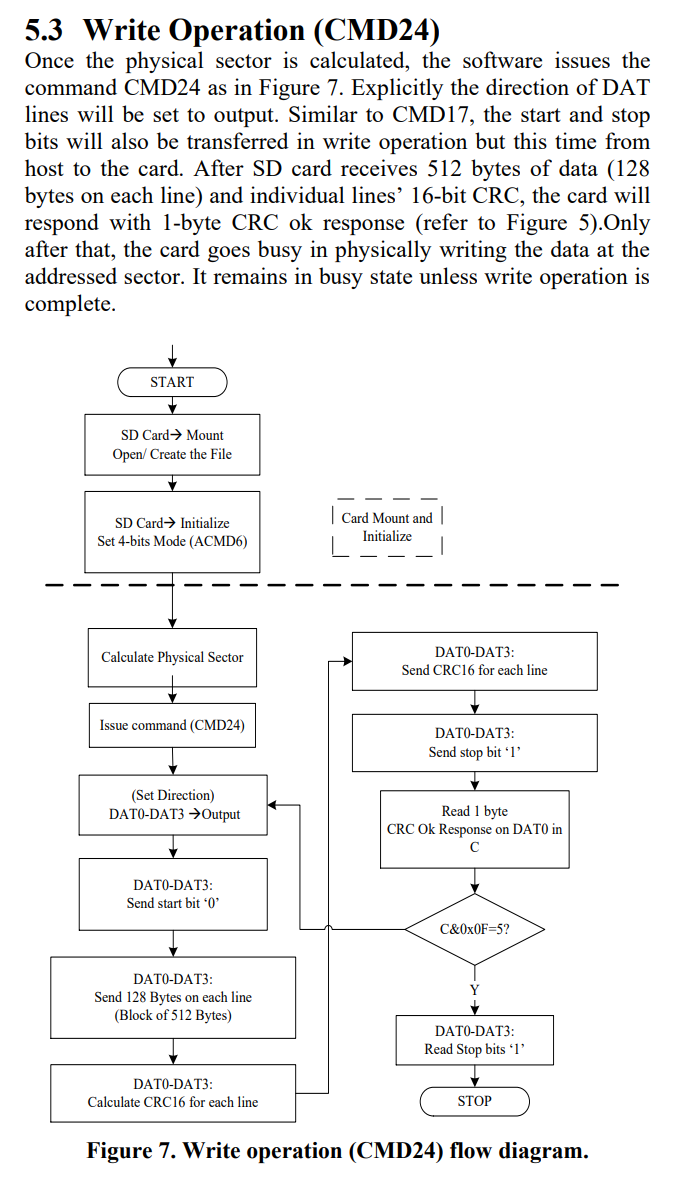
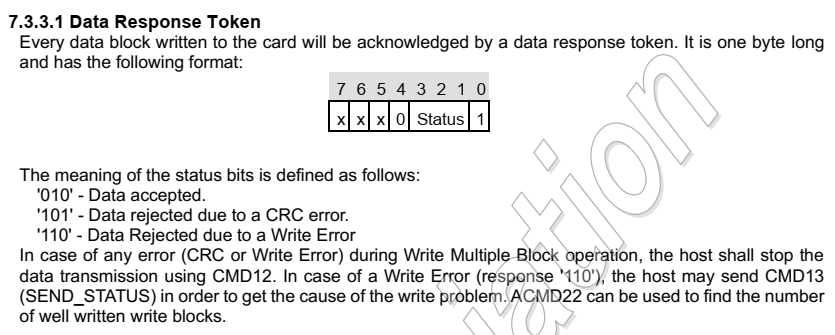
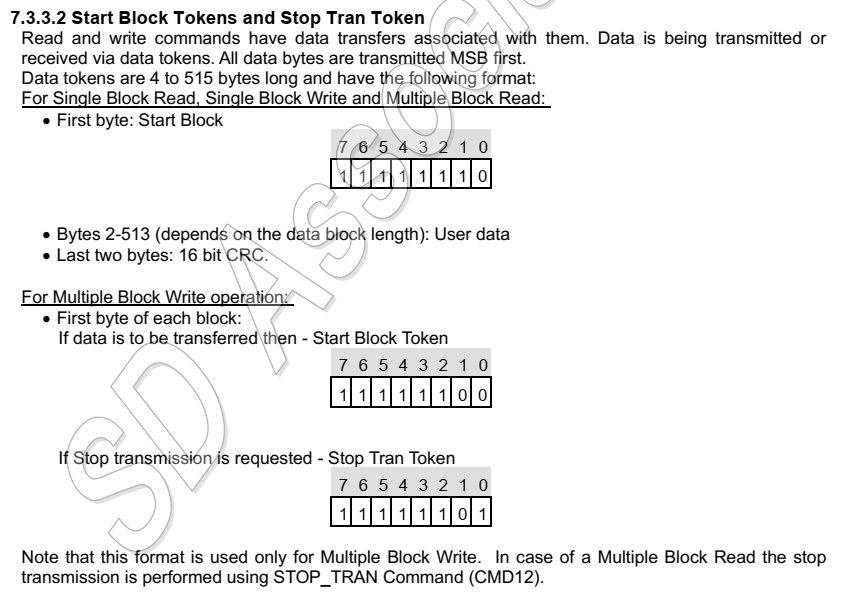
# SDInterface SDWriter Module Design & Development

The SDWriter module needs to closely follow the steps below to ensure a successful write to the SD card:

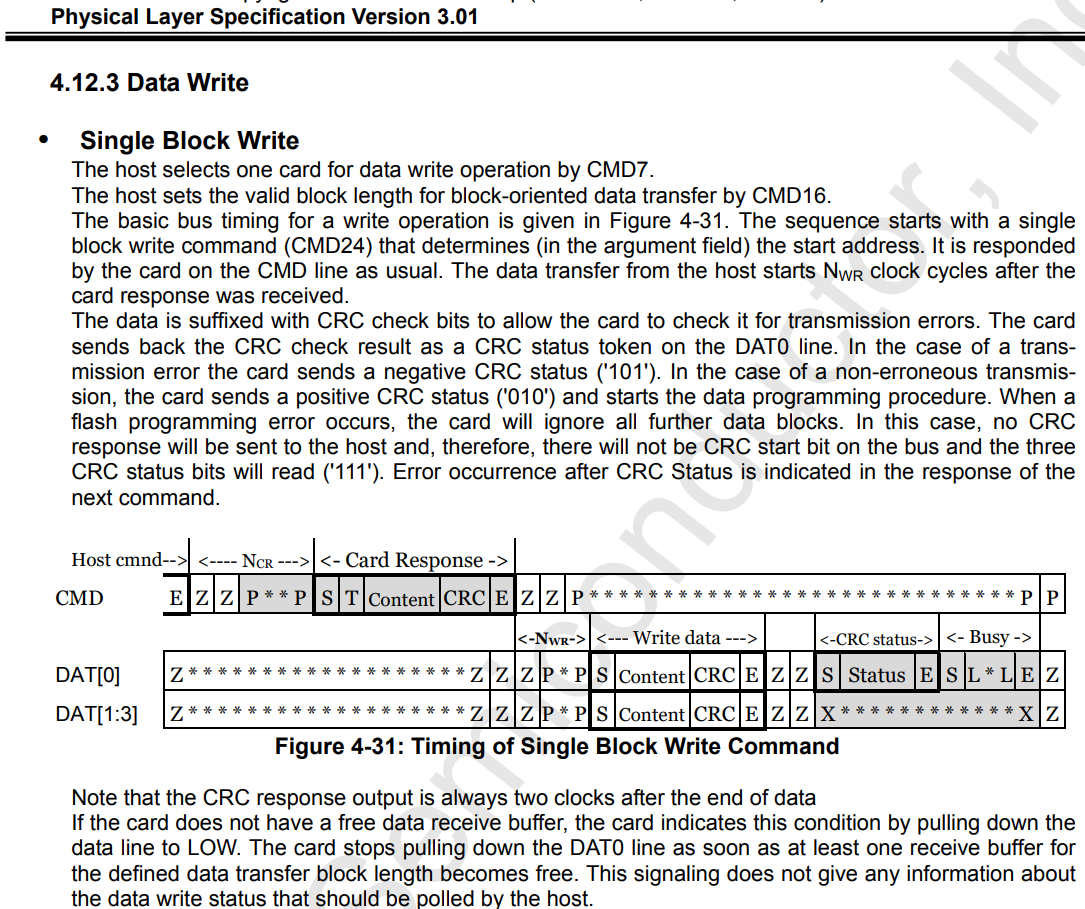
It should be noted that the initialisation sequence will have been completed and the SD card should already be in Transfer Mode as a result of prior RD operations executed as part of the host’s start-up sequence.



Above relates to the CRC OK byte that is transmitted by the SD card on successful receipt of a data block.



The above is important. The first byte sent MUST be the ‘START BLOCK’ byte – 0xFE, but looking at the timings on the next page, it needn’t be a whole byte and we can probably just get away with transmitting a nybble (bits 3-0).



NWR timing specifies a minimum of 2 clocks and no maximum.

## Write Transfer Timings

The *sd\_data\_serial\_host* module uses the following timings are obtained, where data\_cycles = 512\*8 + 8 = 4104:

|  |  |  |  |
| --- | --- | --- | --- |
| Transfer Cycle | State | Action | Effect |
| 1 | WRITE\_DAT | Pull DAT0 LOW | START bit. |
| 2 to 4,105 | WRITE\_DAT | Send data bits & set CRC. | SD card receives 512-byte payload as bits, CRC function records data. |
| 4,105 | WRITE\_DAT | Set crc\_en to 0 | Disables CRC generation/generates CRC. |
| 4,106 to 4,121 | WRITE\_DAT | Send 16 CRC bits. | SD card receives CRC to verify data payload. |
| 4,122 | WRITE\_DAT | DAT0 HIGH. | Send ‘STOP’ bit to SD card. |
| 4,123 | WRITE\_DAT | Disable SD transmission. | End data transmission. Reset SD phy to RD direction. |
| >=4,125 & start\_bit (DAT0 LOW) | WRITE\_CRC | Read first 3 bits of CRC status. | Reads first 3 LSB of CRC OK byte. Move on to next state once 3 bits read. |
| Don’t care | WRITE\_BUSY | Wait for DAT0 to go HIGH. | End of operation. |

The HDL in SDWriter uses different timing, and initiates the data payload bitstream to the SD card with an index of 1, not 0, so widx = 1 when the first bit of the first byte is sent. data\_cycles in SDWriter = 4,096, so this means the SDWriter’s timings differ slightly from those above and look more like the following:

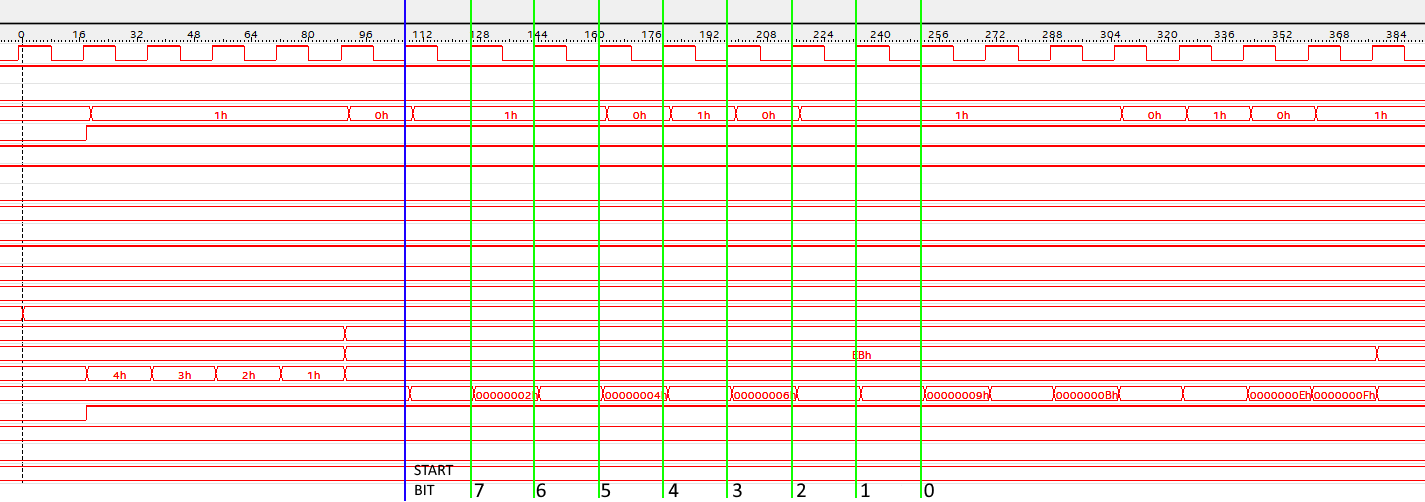
|  |  |  |  |
| --- | --- | --- | --- |
| widx | State | Action | Effect |
| 0 | WRITE\_DAT | Pull DAT0 LOW | Signal to SD card start of transfer. |
| 1 to 4,096 | WRITE\_DAT | Send data bits. | SD card receives 512-byte payload as bits, CRC function records data. |
| 4,096 | WRITE\_DAT | Set crc\_en to 0 | Disables CRC generation/generates CRC. |
| 4,097 to 4,112 | WRITE\_DAT | Send CRC bits. | SD card receives CRC to verify data payload. |
| 4,113 | WRITE\_DAT | DAT0 HIGH. | Send ‘STOP’ bit to SD card. |
| 4,114 | WRITE\_DAT | Disable SD transmission. | End data transmission. Reset SD phy to RD direction. |
| >=4,115 & start\_bit (DAT0 LOW) | WRITE\_CRC | Read first 3 bits of CRC status. | Reads first 3 LSB of CRC OK byte. Move on to next state once 3 bits read. |
| Don’t care | WRITE\_BUSY | Wait for DAT0 to go HIGH. | End of operation. |

It’s clear from the above timing tables that the *sd\_data\_serial\_host* has an additional 8 bits that it sends as part of the payload. I’ve no idea where these bits are sourced from.

I’ve added a stage to the WRITE transaction to check for timeouts whilst waiting for a response from the SD card after sending the payload. The timeout state will send CMD13 to get the Card Status (see pp.115-116 in the Part1\_Physical\_Layer specs). This indicates that a general error is being caught.

### Issues after INIT optimisation

* ~~WRITE process isn’t starting properly – the first byte, 0xEB, is being written twice.~~
* ~~WRITE process isn’t exiting properly – interface isn’t left in a state where a subsequent READ can be performed. Probably not de-selecting the SD card.~~

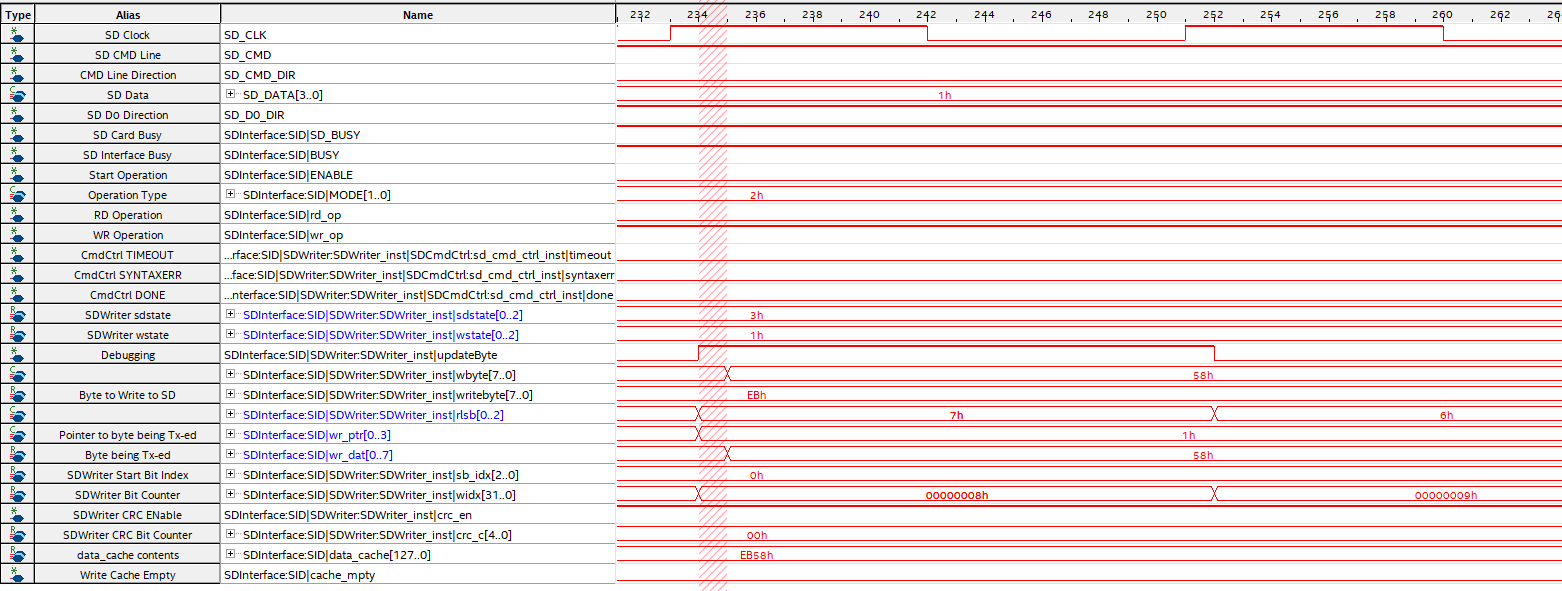


First byte, 0xEB, should end here but instead is being written once more.

Figure 1 - First byte is being written twice.

WRITE process not exiting properly is fixed. I hadn’t finished the HDL in the ENDSEL state, so it wasn’t exiting properly (or de-selecting the card).

Re: WRITE process not starting properly. This appears to be a timing issue, demonstrated in the diagram below, where the byte to written is updated before the byte from the **ByteStreamWriter** is updated due to the new **widx** value.



The red shaded area highlights the issue area – where **updateByte** goes HIGH, the value of **wbyte** is placed in **writebyte**. As can be seen, **wbyte** is still 0xEB at this time, as it doesn’t update to the next byte value (0x58) until a clock later.

The fix for this is to delay the **updateByte** section of code by 1 clock.

It’s worth pointing out that this first byte duplication error was first noticed after the INIT optimisation, but may have been present before this.