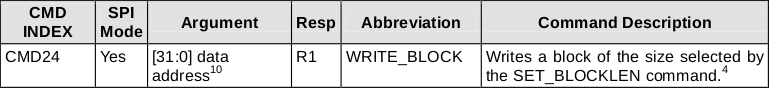
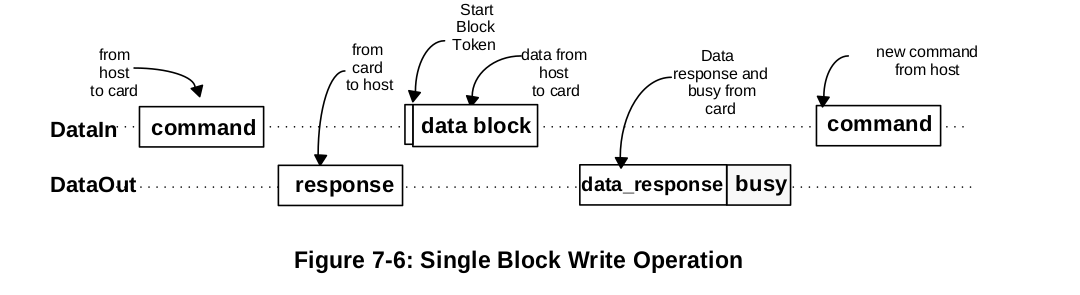
Writing Blocks

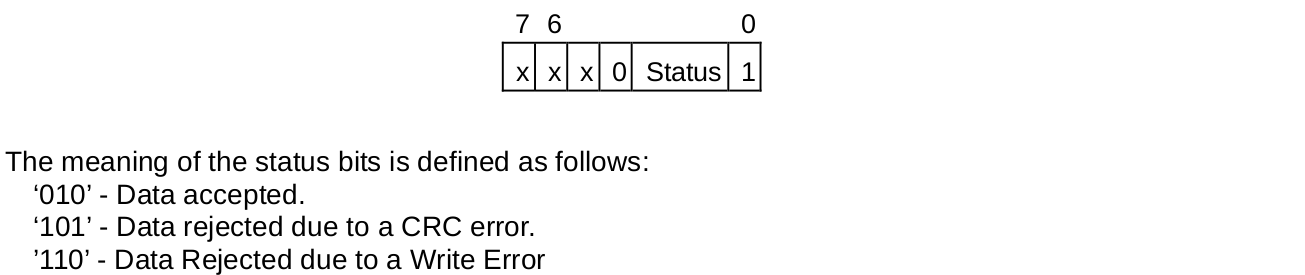
Writing, just like reading, is done in blocks of 512 bytes. The command is CMD24 - WRITE\_BLOCK. The format for this command is shown below.



The flow for a single block write is shown below:



Notice that we send the write block command, wait for a response (R1), then send a start block token (0xFE) followed by 512 bytes of data to be written. After this we will wait for new type of token from the card: a data response token. The format for this is shown below:



If the card accepts the data, we will get the token 0bxxx00101. The card will then send busy tokens (0x00) until it has finished writing the data.

Putting all of this into a function looks something like this:

#**define** CMD24 24

#**define** CMD24\_ARG 0x00

#**define** SD\_MAX\_WRITE\_ATTEMPTS 3907

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Write single 512 byte block

token = 0x00 - busy timeout

token = 0x05 - data accepted

token = 0xFF - response timeout

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**uint8\_t** SD\_writeSingleBlock(**uint32\_t** addr, **uint8\_t** \*buf, **uint8\_t** \*token)

{

**uint8\_t** readAttempts, read;

// set token to none

\*token = 0xFF;

// assert chip select

SPI\_transfer(0xFF);

CS\_ENABLE();

SPI\_transfer(0xFF);

// send CMD24

SD\_command(CMD24, addr, CMD24\_CRC);

// read response

res[0] = SD\_readRes1();

// if no error

**if**(res[0] == SD\_READY)

{

// send start token

SPI\_transfer(SD\_START\_TOKEN);

// write buffer to card

**for**(**uint16\_t** i = 0; i < SD\_BLOCK\_LEN; i++) SPI\_transfer(buf[i]);

// wait for a response (timeout = 250ms)

readAttempts = 0;

**while**(++readAttempts != SD\_MAX\_WRITE\_ATTEMPTS)

**if**((read = SPI\_transfer(0xFF)) != 0xFF) { \*token = 0xFF; **break**; }

// if data accepted

**if**((read & 0x1F) == 0x05)

{

// set token to data accepted

\*token = 0x05;

// wait for write to finish (timeout = 250ms)

readAttempts = 0;

**while**(SPI\_transfer(0xFF) == 0x00)

**if**(++readAttempts == SD\_MAX\_WRITE\_ATTEMPTS) { \*token = 0x00; **break** }

}

}

// deassert chip select

SPI\_transfer(0xFF);

CS\_DISABLE();

SPI\_transfer(0xFF);

**return** res[0];

}

Note that we send the write command to the card and make sure there are no errors in the R1 response:

// send CMD24

SD\_command(CMD24, addr, CMD24\_CRC);

// read response

res[0] = SD\_readRes1();

// if no error

**if**(res[0] == SD\_READY)

{

/\*\*\*/

}

If R1 is error free, we send the start token and then start transmitting the data in the buffer:

// send start token

SPI\_transfer(SD\_START\_TOKEN);

// write buffer to card

**for**(**uint16\_t** i = 0; i < SD\_BLOCK\_LEN; i++) SPI\_transfer(buf[i]);

When we have finished writing the buffer, we wait for the card to send a data response token. Remember data accepted tokens are 0bxxx00101, so we mask the upper three bits of the first non-0xFF response we get and see if it is equal to 0b00000101:

// wait for a response (timeout = 250ms)

readAttempts = 0;

**while**(++readAttempts != SD\_MAX\_WRITE\_ATTEMPTS)

**if**((read = SPI\_transfer(0xFF)) != 0xFF) { \*token = 0xFF; **break**; }

// if data accepted

**if**((read & 0x1F) == 0x05)

{

/\*\*\*/

}

Finally, we wait for the card to finish writing the data:

// wait for write to finish (timeout = 250ms)

readAttempts = 0;

**while**(SPI\_transfer(0xFF) == 0x00)

**if**(++readAttempts == SD\_MAX\_WRITE\_ATTEMPTS) { \*token = 0x00; **break** }

Let's try this out, simply writing a set block of data to a sector on the card. Here I will fill the buffer with the value 0x55 and write to address 0x00000100:

// fill buffer with 0x55

**for**(**uint16\_t** i = 0; i < 512; i++) buf[i] = 0x55;

// write 0x55 to address 0x100 (256)

res = SD\_writeSingleBlock(0x00000100, buf, &token);

Timeouts

As with read operations, we need to set a timeout value so our program does not get hung up if the card never responds. Per section 4.6.2.2 of the physical spec. *Maximum length of busy is defined as****250****ms for all write operations.*

Set the maximum write attempts to:

*(0.25s \* 16000000 MHz)/(128 \* 8 bytes) = 3907*

This should give just over 250ms. Note that there are 4 cases with this function that we should check after we call it:

* R1 != 0x00 → Error writing block (parse R1 for details)
* R1 == 0x00 and token == 0x05 → Success
* R1 == 0x00 and token == 0x00 → Busy Signal timeout
* R1 == 0x00 and token == 0xFF → No response after R1

Based on these combinations, we know whether the write was successful or if we need to try again.