Работа с SDC/MMC на примере драйвера под Z-Controller (Z80)

Лет 7 все собирался написать свой собственный драйвер, для работы с **SDC/MMC** под **Z80**, да всё как-то не складывалось. В своих проектах все это время использовал драйвер от **Savelij**'а. В принципе меня таковой вполне устраивал, т. к. недостатки в оном не сильно критичны. Драйвер сделан весьма упрощённо и соответственно не работает с некоторыми картами, но при этом очень компактен. Так же, из-за специфики реализации, работает с картами медленнее, чем можно было бы (что, пожалуй, вообще не критично). В свое время Слава меня таки спас от написания драйвера, за что ему мега респект.

И вот, в начале 2015 года , удалось поймать настроение и время. И, наконец-то, написать свой драйвер. После не очень долгих экспериментов, таковой был доработан и встроен в основные, на тот момент, проекты, а конкретно: **WildCommander**, **vDOS** и **ts-bios**. В старичке **WDC** таковой думаю обновить в обозримом будущем, ибо есть определенные сложности с организацией памяти. А вот в проектах-трупах типа **WildPlayer**'а и всяких **SD** тестерах вряд-ли что-либо стану менять...

Отдельная благодарность **elm-chan**'у, за отличнейшую статью про работу с **SDC/MMC**, со множеством иллюстраций и комментариев. А так же, за выложенные исходники драйверов под разные архитектуры. Разбор работы с картами делал по его Си исходникам, с оглядкой на спецификации и статью...

Немного про SDC/MMC карты

Вкратце расскажу про Secure Digital Memory Card (далее **SDC**) и Multi Media Card (далее **MMC**). Внутри каждой карты находится микроконтроллер, который занимается управлением внутренней флеш-памятью (контролем износа, коррекцией ошибок, всякими другими преобразованиями). Соответственно снаружи карта выглядит как обычный внешний носитель на подобии HDD, обмен с которым происходит в виде чтения/записи 512 байтовых блоков (секторов).

Сами карты могут подразделяться на различные форм-факторы, такие как **RS-MMC**, **miniSD** и **microSD**, что влияет на их размер, но никак не на функциональность.

Особенности работы с носителями на основе флеш-памяти

SDC/MMC карты имеют как преимущества, так и недостатки. Основное преимущество — это отсутствие задержек при обращении к данным, находящимся разрозненно на носителе. Из недостатков можно отметить достаточно быстрый износ памяти, т. к. ресурс таковой весьма ограничен.

Следует обратить особое внимание на то, как работает запись на носителях на основе флеш-памяти, коими являются все карты. Для того, чтобы записать что-либо - нужно вначале стереть блок с данными, в который будет идти запись. Как правило память делится на блоки размером от 16кб и выше, так называемые **erase blocks** в NAND Flash. Естественно, все эти сложные операции выполняет контроллер карты. Но, зная как это работает, можно серьезно увеличить скорость записи!

Так же нужно учитывать, что при много-секторной записи карта временно убивает все данные в блоке в который идет запись, и восстанавливает их только после завершения записи, либо при переходе в следующий блок (размер блока начинается от 16кб, посему можно угробить данные, которые лежат далеко перед или после сектора, который в данный момент пишется).

В одно-секторной записи это тоже происходит, но блок восстанавливается сразу после завершения передачи данных сектора карте, что сильно повышает износ карты и замедляет скорость работы. Посему не советую использовать таковую.

Работа через SPI и реализация такового в Z-Controller

SPI (Serial Peripheral Interface) представляет из себя последовательный синхронный интерфейс передачи данных в режиме полного дуплекса (данные передаются одновременно в обе стороны).

При работе с **SDC/MMC** по **SPI** используются сигналы:
DI (Data In) — данные уходящие на карту
DO (Data Out) — данные приходящие с карты
SCLK (Serial Clock) — тактовый сигнал, служит для передачи тактового сигнала контроллеру карты
CS (Chip Select) — выбор карты

В дальнейших примерах кода под **Z80**, для работы с **SDC/MMC** используются порты по стандарту **Z-Controller**'а (далее **ZC**). Особо подробно реализацию такового здесь я затрагивать не стану. Со всей этой информацией можно ознакомиться по ссылкам ниже.

Для работы с картами по **SPI** в **ZC** используется 2 порта ввода-вывода. Порт конфигурации (#77) и порт данных (#57).

порт **#77** на запись:
bit 0 – питание карты (0 — выключено, 1 — включено) [на ZX Evolution не реализовано]
bit 1 – управление сигналом CS (выбор карты)

порт **#77** на чтение:
bit 0 – если 0 — карта установлена, 1 — карта отсутствует
bit 1 – если 1 — то на карте стоит защита от записи

порт **#57** используется как на запись, так и на чтение, для обмена данными по **SPI**. Тактирование осуществляется автоматически, при записи/чтении любого значения в/из порта **#57**. При этом формируются 8 тактовых импульсов на выходе **SCLK**.

В принципе, реализация **SPI** на других контроллерах под **ZX** отличается только адресами портов и, возможно, битами порта конфигурации, что весьма легко изменить в драйвере.

Более подробно про **SPI** можно почитать здесь: http://elm-chan.org/docs/spi_e.html
Документация на **Z-Controller**: http://pentagon.nedopc.com/ZC.rar

Отправка команды и ожидание ответа

Карта готова принять команду, когда оная выставляет линию **DO** в 1, т. е. от карты приходят #FF. Перед подачей команды, сигнал **CS** нужно выставить в 1, а затем в 0. И держать его так в течении всей транзакции (отправка команды, получение ответа, а также передачи данных, если есть).

Команды на карту подаются в виде пакетов, размером в 6 байт. Через 0-8 байт (в случае **SDC**), либо через 1-8 байт (в случае **MMC**) карта возвращает ответ, тип которого зависит от команды, рассмотрим самые основные.

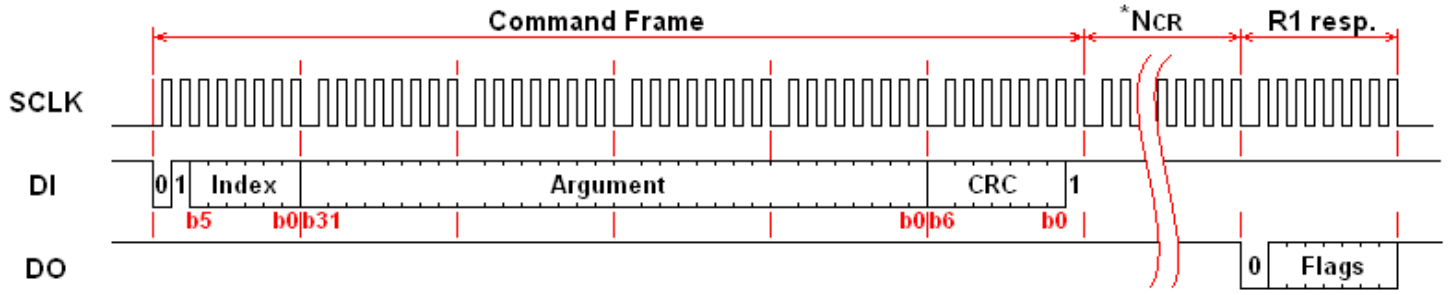
R1 (%0xxxxxxx)

bit 7 – всегда в 0
bit 6 – Parameter Error (к команде указаны недопустимые параметры)
bit 5 – Address Error (указан недопустимый адрес блока/сектора)
bit 4 – Erase Sequence Error (ошибка в последовательности команд стирания)
bit 3 – Command CRC Error (не соответствие контрольной суммы и полученного пакета команды)
bit 2 – Illegal Command (не поддерживаемая команда)
bit 1 – Erase Reset (возникает если выйти за пределы предварительно стертого блока, при записи)
bit 0 – In Idle State (карта находится в режиме инициализации)

R1b – тот же R1, только данный ответ дается на команды, которые выполняются дольше стандартного времени. Сопровождается выставлением флага busy (на DO подается низкий уровень, пока выполняется команда)

R3 (R1 + OCR(32bit)) – данный ответ возвращается при подаче команды на чтение регистра OCR (Operation Condition Register). Расписывать подробно оный здесь не стану, т. к. не вижу в этом особой необходимости.

Схема отправки команды и приема ответа



Как видно из рисунка выше, есть 2 линии данных (**DI** и **DO**). Передача по ним идет синхронно и одновременно в обе стороны. Запись/чтение очередного байта инициирует 8 тактирований сигнала **SCLK** и соответственно передачу по 8 бит в каждую сторону. При чтении из порта данных оные забираются из буфера, в ответ полученному байту карте уходит #FF . При записи же байта в порт данных оный передается на линию **DI** начиная от старшего бита к младшему.

Основные команды поддерживаемые картами в режиме SPI

Команда	Аргумент	Ответ	Данные	Описание
CMD0		R1	нету	Программный сброс
CMD1		R1	нету	Запуск процесса инициализации
ACMD41(*1)	*2	R1	нету	Запуск процесса инициализации (Только для SDC)
CMD8	*3	R7	нету	Поддерживаемые напряжения (Только для SDCv2)
CMD9		R1	есть	Получить CSD
CMD10		R1	есть	Получить CID
CMD12		R1b	нету	Прекратить читать данные
CMD16	длина блока[31:0]	R1	нету	Изменение размера блока на чтение и запись
CMD17	адрес[31:0]	R1	есть	Загрузить блок (сектор)
CMD18	адрес[31:0]	R1	есть	Загрузить несколько блоков (секторов)
CMD23	кол-во блоков[15:0]	R1	нету	Задать кол-во блоков, которое будет загружено/записано (Только для MMC)
ACMD23(*1)	кол-во блоков[22:0]	R1	нету	Задать кол-во блоков на стирание, при след. команде CMD25 (Только для SDC)
CMD24	адрес[31:0]	R1	есть	Записать блок (сектор)
CMD25	адрес[31:0]	R1	есть	Записать несколько блоков (секторов)
CMD55(*1)		R1	нету	Команда префикс, перед ACMD<n>
CMD58		R3	нету	Получить OCR

*1:ACMD<n> подразумевает последовательность команд из CMD55 + CMD<n>.
*2: Rsv(0)[31], HCS[30], Rsv(0)[29:0]
*3: Rsv(0)[31:12], поддерживаемое напряжение(#01)[11:8], #AA[7:0]

Инициализация и определение SDC/MMC карт

Для работы с картой по интерфейсу **SPI**, нужно активировать соответствующий режим во время инициализации карты. При подаче питания на карту, таковая входит в свой обычный режим, для включения режима работы по **SPI** нужно выполнить следующую последовательность:

1. после подачи питания на карту, подождать не менее 1 мс
2. **SPI Clock** переключить на частоту 100-400кГц (если это возможно)
3. выставить **DI** и **CS** в 1 на 74 импульса тактирований по **SCLK** (в случае с **ZC** надо подать не менее 10 кодов #FF в сторону карты)
4. выставляем **CS** в 0 и посылаем на карту **CMD0** с правильной **CRC**. В ответ должен прийти статус R1 с выставленным битом In Idle State (#01)
5. т. к. **CS**=0, то карта переключится в режим работы по **SPI** (проверка **CRC** автоматически отключается).

Алгоритм определения типа и наличия SDC/MMC карты

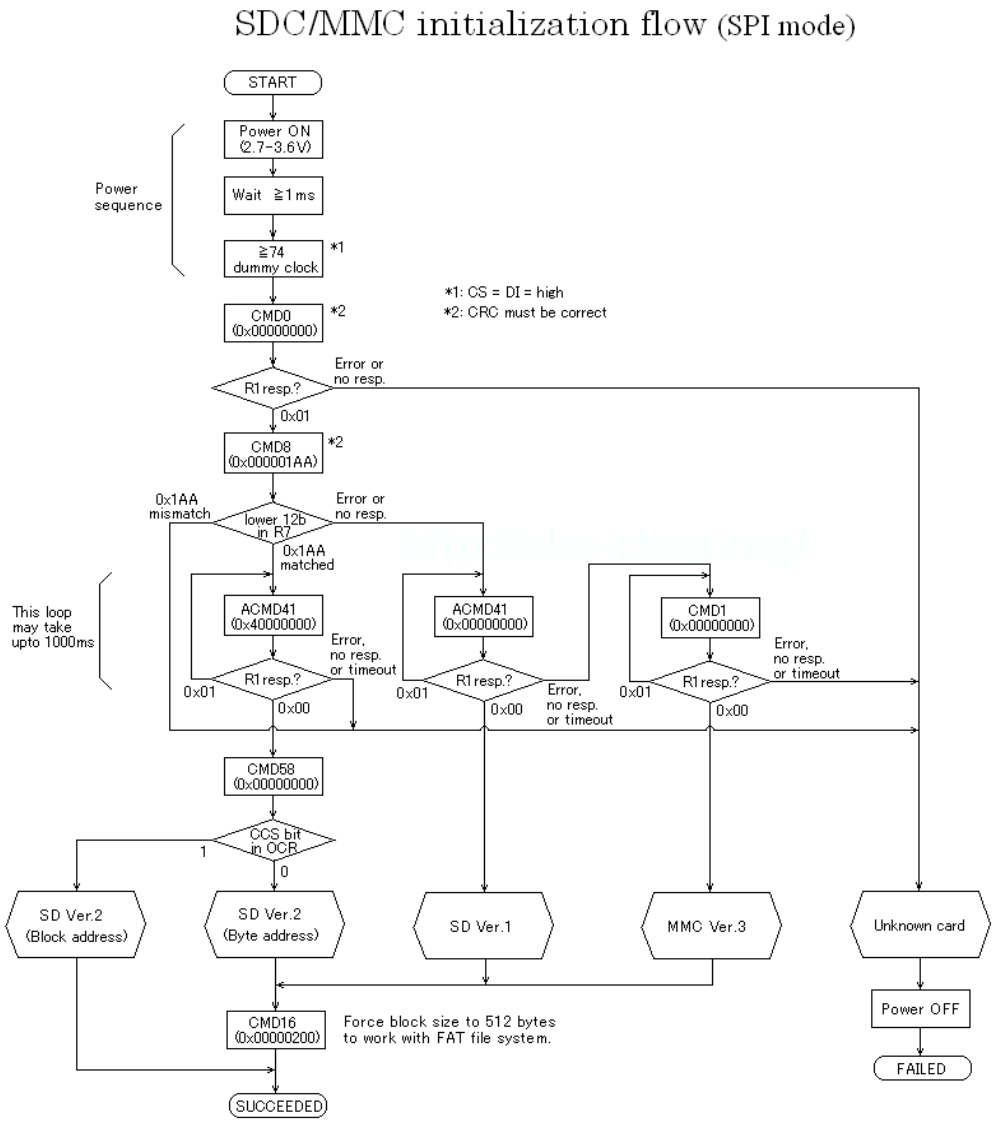
Собственно после включения на карте режима работы по SPI, нужно провести её инициализацию и определить версию и тип карты, т. к. от этого зависит как мы с ней в дальнейшем будем работать!

Каждый шаг инициализации следует пытаться проделывать неоднократно, т. к. карта не всегда с первого раза, а зачастую и не с десятого, отвечает нужным образом. В драйвере из примера у меня выставлено по 8к попыток и, как показала практика, это весьма оправданно.

Опираясь на то, какие команды поддерживают различные виды карт и то, как они на них реагируют, можно определить точно, что за карта перед нами. И соответственно сделать все необходимые приготовления.

К примеру, команду **CMD8** поддерживают только **SD** карты версии 2 и, соответственно, на подачу таковой ошибку выдадут как **SD** карты версии 1, так и **MMC** карты.

На блок-схеме процесс полной инициализации выглядит примерно так (позаимствована у elm-chan'a!):



Процедура определения наличия и инициализации SDC/MMC (Z80)

```
;на выходе: A = 0 - инициализация прошла успешно, A = 1 — карта не поддерживается/не обнаружена
SD_INIT CALL CSH
        LD DE,512+10
        CALL CYCL;выгребаем 512+10 байт, на всякий случай, мало ли карта что-то не отдала/приняла

        LD DE,8000;8к циклов ожидания
SDWT    DEC DE
        LD A,D
        OR E
        JP Z,NOSD
        CALL CMD0;команда на инициализацию карты
        JR NZ,SDWT;висим в цикле, пока не будет получен cmd. Resp. (в отведенные 80 тактирований)
        DEC A
        JP NZ,SDWT;висим в цикле, пока не получим ответ без ошибок и с выставленным Idle State флагом (bit 0)

;если все прошло успешно, то карта переключается в SPI режим

        CALL CMD8;запрос на поддерживаемое напряжение питания (только для SDCv2)
        PUSH AF
        IN E,(C)
        IN E,(C)
        IN H,(C)
        IN L,(C)
        POP AF
        JR NZ,SDWT;cmd. Resp. получен?
        BIT 2,A
        JR Z,SDNEW;если нет ошибки Illegal command (бит 2), то перед нами SDCv2
;-------
;cmd08 не поддерживается, соотв. перед нами либо SDCv1, либо MMC
SDOLD   LD DE,8000;8к циклов ожидания
AA      DEC DE
        LD A,D
        OR E
        JR Z,LC;если время вышло, пробуем определить карту как MMC
        LD H,0;HCS в 0
        CALL ACMD41;команда на инициализацию SDC
        JR NZ,AA;ждем cmd. Resp.
        CP 1
        JR Z,AA;ждем выхода из Idle State
        OR A
        JR NZ,LC;если есть какая-либо ошибка, то пробуем определить карту как MMC

;SDv1 Detected
        JR FBS
;-------
LC      LD DE,8000
OO      DEC DE
        LD A,D
        OR E
        JR Z,NOSD;карты нету, либо неизвестный тип
        CALL CMD1;команда на инициализацию MMC
        JR NZ,OO;ждем cmd. Resp.
        CP 1
        JR Z,OO;ждем выхода из Idle State
        OR A
        JR NZ,NOSD;карта не опознана

;MMC Ver.3 Detected
        JR FBS
;-------
SDNEW   LD DE,#01AA
        OR A
        SBC HL,DE
        JR NZ,NOSD;карта не опознана

        LD DE,8000
YY      DEC DE
        LD A,D
        OR E
        JR Z,NOSD
        LD H,#40;HCS в 1
        CALL ACMD41;команда на инициализацию SDC
        JR NZ,YY;ждем cmd. Resp.
        CP 1
        JR Z,YY;ждем выхода из Idle State
        OR A
        JR NZ,NOSD;карта не опознана

;SDv2 Detected
        CALL CMD58;определяем тип адресации (блоковая либо байтовая)
        JR NZ,NOSD
        LD BC,DATA
        IN A,(C)
        IN L,(C)
        IN L,(C)
        IN L,(C)
        BIT 6,A
        JR Z,FBS;SDV2 Byte Addr

;SDv2 Block Address
        LD A,1
SDFND   ;карта успешно опознана
        LD (BLKT),A;выставляем переменную типа адресации (блоковая/байтовая)
;используется для выставления правильного адреса при позиционировании (умножение на 512, если адресация байтовая)
        XOR A
        JP CSH
;-------
FBS     ;выставляем размер блока в 512байт, т. к. карта использует байтовую адресацию
        CALL CMDi6
        JR NZ,NOSD;если не получен cmd. Resp., то считаем, что карты нету
        OR A
        JR Z,SDFND
;-------
NOSD    ;карта не опознана, вылетаем с ошибкой
        CALL SDOFF;выключаем питание (на ZX Evolution не реализовано)
        LD A,1
        OR A
        RET
;-------
CYCL    ;выгреб N байт из SPI
        LD BC,DATA
CY      LD A,#FF
        OUT (C),A
        DEC DE
        LD A,D
        OR E
        JR NZ,CY
        RET
```

Передача данных

При работе с SDC/MMC картами есть 2 типа команд на загрузку и запись данных, одно-блоковые и много-блоковые. Для более эффективной работы лучше использовать последние, т. к. уменьшается количество обращений к карте.

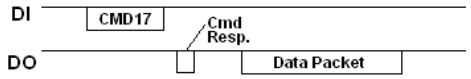
Формат блока данных при записи/чтении:
Data Token (1 байт) + Data Block (512 байт) + CRC (2 байта)

Data Token = %11111110 (для CMD17/18/24)
Data Token = %11111100 (для CMD25)
Data Token = %11111101 (для прекращения передачи данных, после CMD25)

Error Token (%000xxxxx)
bit 4 – Card is locked
bit 3 – Out of range
bit 2 – Card ECC failed
bit 1 – CC Error
bit 0 – Error

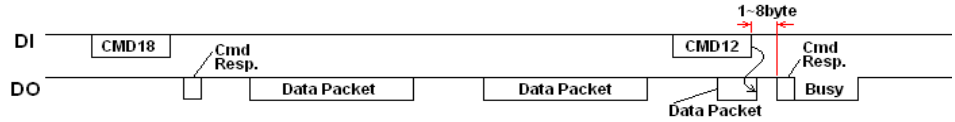
Data Response (%0xxx1)
%010 – Data Accepted
%101 – CRC Error
%110 – Write Error

Схема одно-блокового чтения (CMD17):



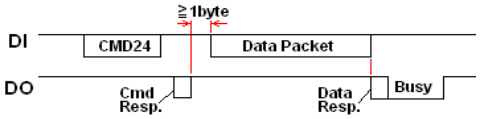
CMD17 в качестве аргумента имеет номер блока (сектора) который надо считать, при формировании такового нужно учитывать тип адресации, который был определен при инициализации карты. Если во время чтения произошла ошибка, то будет возвращен **Error Token** заместо **Data Packet**.

Схема много-блокового чтения (CMD18):



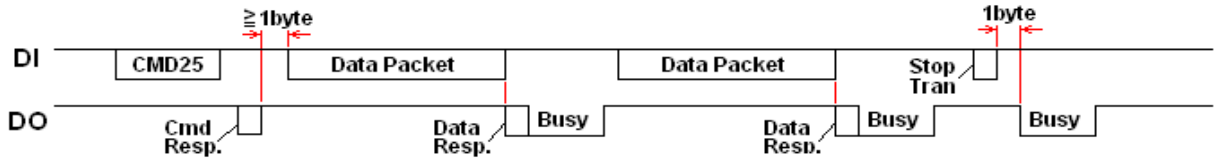
CMD18 так же в качестве аргумента имеет номер блока (сектора) с которого будет начато чтение последовательности. Остановка чтения обуславливается отправкой **CMD12,** после принятия последнего пакета с данными. При этом, первый байт полученный сразу за **CMD12** нужно игнорировать, т. к. он является частью очередного пакета с данными!
В ситуации, когда перед **CMD18** была вызвана **CMD23** (задающая количество блоков, которое будет загружено/записано), чтение прерывается после принятия заданного количества блоков (актуально только для MMC!).

Схема одно-блоковой записи (CMD24):



CMD24 во многом аналогична **CMD17**, сразу после отправки команды и получения **cmd. Resp.** можно посылать на карту пакет данных, предварительно дождавшись пока карта выставит DO в 1. Заместо CRC можем записывать любое значение, если мы не включили проверку оной. Когда пакет с данными отправлен, то сразу же за ним будет Data Response. Следом за Data Response идет флаг busy, нужно дождаться, пока карта снимет оный.

Схема много-блоковой записи (CMD25):



CMD25 запускает запись нескольких блоков идущих друг за другом, начиная с указанного адреса. Если количество передаваемых блоков не было указано, то оная начнется как много-блоковая запись, с завершением при подаче токена **Stop Tran**. Следом картой выставляется флаг busy (на DO подается низкий уровень, пока карта занята). Для **SDC** много-блоковая запись должна быть завершена токеном Stop Tran, в независимости от того, было ли задано количество блоков или нет.

Процедуры чтения и записи (много-блоковые)

```
RDDSE   ;загрузка блока секторов
;        i:[BC,DE] – номер сектора (в BC старшая часть), номер от 0
;                HL – адрес куда грузить
;                 A – количество секторов (512б)

        EXA
        CALL CMD18
        JR NZ,$
        EXA
RD1     EXA
        CALL WTDO
        CP #FE
        JR NZ,$-5;ожидаем токен начала данных (%11111110), следом за ним начинается принимаемый сектор (512байт + 2байта crc)
        CALL READS;принимаем 512б данных и пропускаем crc
        EXA
        DEC A
        JR NZ,RD1

        CALL CMD12
        CALL SNB
        CALL WTBY
        JP CSH

READS   PUSH BC,DE

;читаем непосредственно сам сектор (512б)
        LD BC,DATA
        LD D,32
RZ1
.16     INI;тираж строки 16 раз
        DEC D
        JP NZ,RZ1

        LD BC,DATA
        IN A,(C);выгребаем 2 байта crc
        IN A,(C)
        POP DE,BC
        RET
;---------------------------------------

SDDSE   ;запись блока секторов
;        i:[BC,DE] – номер сектора
;                HL – адрес в памяти
;                 A – количество секторов (512б)

        EXA
        XOR A
        IN A,(CONF)
        AND 2
        RET NZ;если запись запрещена, то ничего не пишем на носитель...

        CALL CMD25;даем команду на много-секторную запись
        CALL WAIT;ожидаем, пока контроллер карты будет готов принять пакет с данными
        EXA
SD1     EXA
        CALL SAVDS;передача блока данных, с токеном начала + 512 байт данных и 2 байт crc
        CALL DRESP;ожидание Data Resp. и снятия busy
        EXA
        DEC A
        JR NZ,SD1

SDND    LD BC,DATA
        LD A,#FD;токен конца много-секторной операции (!)
        OUT (C),A
        CALL SNB
        CALL WTBY
        JP CSH

SAVDS   PUSH BC,DE
        LD BC,DATA
        LD A,#FC;токен начала блока данных
        OUT (C),A

        LD D,32
SV1
.16     OUTI;тираж строки 16 раз
        DEC D
        JP NZ,SV1

        LD BC,DATA
        LD A,#FF
        OUT (C),A;посылаем 2 байта crc (любой)
        OUT (C),A
        POP DE,BC
        RET
```

Основные функции и константы

```
CONF    EQU #77;порт конфигурации
DATA    EQU #57;порт данных

CMD_1   EQU %01000000+1; инициализация
CMD_12  EQU %01000000+12;остановка передачи
CMD_16  EQU %01000000+16;задание размера блока
CMD_18  EQU %01000000+18;много-секторное чтение
CMD_25  EQU %01000000+25;много-секторная запись
ACMD_41 EQU %01000000+41;инициализация (только SDC)
CMD_55  EQU %01000000+55;команда префикс к ACMD
CMD_58  EQU %01000000+58;чтение OCR

CMD00   DB %01000000+0:DS 4:DB #95;программный сброс
CMD08   DB %01000000+8,0,0,1,#AA,#87;напряжение карты (только для SDCv2)
CMD16   DB %01000000+16,0,0,2,0,#FF;задание размера блока (512б)

BLKT    NOP; тип адресации (0 — байтовая, 1 — блоковая)

CSH     ;выставление CS в 1
        PUSH BC,AF
        LD BC,CONF
        LD A,%00000011
        OUT (C),A
        LD BC,DATA
        LD A,#FF
        OUT (C),A
        POP AF,BC
        RET

CSL     ;выставление CS в 0
        PUSH BC,AF
        LD BC,CONF
        LD A,%00000001
        OUT (C),A
        LD BC,DATA
        LD A,#FF
        OUT (C),A
        POP AF,BC
        JP WAIT

SNB     ;пауза в 128 тактирований
        PUSH BC,AF
        LD B,16
SnB     XOR A
        IN A,(DATA)
        DJNZ SnB
        POP AF,BC
        RET
;-------
CMDo    CALL CSH
        CALL CSL
CMDx    PUSH BC
        LD BC,DATA
        OUT (C),A
        XOR A
        OUT (C),A
        OUT (C),A
        OUT (C),A
        OUT (C),A
        DEC A
        OUT (C),A
        POP BC
        RET

CMD1    ;запуск инициализации (только для MMC)
        LD A,CMD_1
        CALL CMDo
        JP RESP

CMD12   ;завершение передачи данных (вызывается при завершении много-секторного чтения)
        LD A,CMD_12
        CALL CMDx
        XOR A
        IN A,(DATA)
        JP RESP

CMD55   ;вызывается перед ACMD<n>
        LD A,CMD_55
        CALL CMDo
        JP RESP

CMD58   ;чтение OCR
        LD A,CMD_58
        CALL CMDo
        JP RESP
;-------
ACMD41  ;запуск инициализации (только для SDC)
        CALL CMD55
        CALL CSH
        CALL CSL

        LD BC,DATA
        LD A,ACMD_41
        OUT (C),A
        LD L,0
        OUT (C),H
        OUT (C),L
        OUT (C),L
        OUT (C),L
        DEC L
        OUT (C),L
        JP RESP
;-------
CMD18   ;запуск много-секторного чтения (завершение производится по cmd12)
        LD A,CMD_18
CMDz    CALL CSH
        CALL CSL
        PUSH HL,DE,BC

        LD L,C
        LD H,B

        LD C,A
        LD A,(BLKT)
        OR A
        JR NZ,CMzz
        EX DE,HL
        ADD HL,HL
        EX DE,HL
```

```
        ADC HL,HL
        LD H,L
        LD L,D
        LD D,E
        LD E,A

CMzz    LD A,C
        LD BC,DATA
        OUT (C),A
        OUT (C),H
        OUT (C),L
        OUT (C),D
        OUT (C),E
        LD A,#FF
        OUT (C),A
        POP BC,DE,HL
        JP RESP

CMD25   ;запуск много-секторной записи (завершение производится по стоп токену!)
        LD A,CMD_25
        JR CMDz
;-------
CMDi6   ;задание размера блока (должно быть всегда 512б)
        LD HL,CMD16
        JR CMD

CMD8    ;узнать какое напряжение нужно карте (только для SDCv2)
        LD HL,CMD08
        JR CMD

CMD0    ;программный сброс карты
        LD HL,CMD00
CMD     CALL CSH
        CALL CSL
        LD BC,DATA
        OUTI
        OUTI
        OUTI
        OUTI
        OUTI
        OUTI

RESP    ;ожидание cmd. Resp. (в течении 80 тактирований)
        PUSH DE,BC
        LD BC,DATA
        LD D,10
RESp    IN A,(C)
        BIT 7,A
        JR Z,REZ
        DEC D
        JR NZ,RESp
        INC D
REZ     POP BC,DE
        RET
;-------
DRESP   ;ожидание Data Response
        PUSH BC,AF
        LD BC,DATA
        IN A,(C)
        CP #FF
        JR Z,$-4
        AND #1F
        CP 5
        JR NZ,ER;если Data Resp. принят без ошибок, то идем дальше (%0XXX1, xxx - биты статуса [%010 - данные приняты])
        IN A,(C)
        OR A
        JR Z,$-3;ждем пока пройдет busy
        POP AF,BC
        RET

ER      ;подсвечиваем бордер кодом ошибки и вешаемся (предварительно дав токен конца данных, дабы не гробилась пага целиком)
        AND 7
        OUT (254),A
        CALL SDND;даем стоп токен
        CALL SDOFF;вырубаем питание карты
        JR $;здесь можно сделать вылет по ошибке, с сообщением, что носитель требует замены
;-------
WTBY    ;ожидание снятия BUSY
        PUSH BC,AF
        LD BC,DATA
        IN A,(C)
        OR A
        JR Z,$-3
        POP AF,BC
        RET

WTDO    ;ожидание DO
        PUSH BC
        LD BC,DATA
        IN A,(C)
        CP #FF
        JR Z,$-4
        POP BC
        RET

WAIT    ;ожидаем пустой шины
        PUSH BC,AF
        LD BC,DATA
        IN A,(C)
        INC A
        JR NZ,$-3
        POP AF,BC
        RET
;-------
SDOFF   ;выключаем питание карты
        XOR A
        OUT (CONF),A
        OUT (DATA),A
        RET
```

Полезные ссылки

Все иллюстрации, использованные в этой статье, взяты отсюда: http://elm-chan.org/docs/mmc/mmc_e.html

Про SPI можно почитать здесь: http://elm-chan.org/docs/spi_e.html
Неплохая статейка про SD карты: http://www.chlazza.net/sdcardinfo.html
Спецификации на SDC: https://www.sdcard.org/downloads/pls/
Документация на Z-Controller: http://pentagon.nedopc.com/ZC.rar

Исходник оригинала драйвера, фрагменты которого приведены в статье:
https://zx-evo-fpga.googlecode.com/hg/pentevo/soft/WC/source/DSDZC.ASM
Процедуры с одно-блоковыми чтением и записью можно посмотреть тут:
https://zx-evo-fpga.googlecode.com/hg/pentevo/soft/WC/source/plugins/mounter/DMN_SDZ.ASM