Z-CONTROLLER

Алексей Жабин [koe/ТОГЯФ]
Вячеслав Савенков [savelij]

Устройство представляет собой универсальный периферийный контроллер для ZX-Spectrum – совместимых компьютеров, оснащенных системной шиной ZX-BUS. В рамках контроллера реализована поддержка PS/2 мышки и клавиатуры, IDE-интерфейса и карты flash-памяти (SD, подключается по SPI-интерфейсу). При работе с портами ввода-вывода контроллера центральный процессор компьютера не приостанавливается для выполнения пустых тактов ожидания [wait], благодаря чему исключается возможность нарушения привязки к реальному масштабу времени в критичных программах (видеоэффекты, проигрывание цифровой музыки, работа с ЦАП/АЦП).

В предлагаемой вниманию читателя статье рассматривается аппаратная реализация узлов контроллера и описывается методика работы с SD-картой с помощью используемого в устройстве интерфейса SPI.

Внешний вид контроллера показан на рис.1. Устройство реализовано в виде периферийной платы, устанавливаемой в один из слотов системной шины ZX-BUS. Конструктив контроллера разрабатывался прежде всего с ориентацией на его использование совместно с компьютером “Pentagon-1024sl” версии 2.2 и, как и конструктив компьютера, ориентирован на AT-корпус. Для удобства подключения при использовании другого типа компьютера или корпуса предусмотрены отдельные штырьковые 5-контактные соединители для клавиатуры и мышки, разводка контактов в которых соответствует разводке контактов в аналогичных соединителях на материнских платах компьютеров IBM PC.

Рис.1. Внешний вид контроллера

Рассмотрим вначале аппаратную часть контроллера. Его принципиальная электрическая схема показана на рис.2. В основе схемы лежит связка между микроконтроллером КР1878ВЕ1 (производства ОАО “Ангстрем”, г. Зеленоград) DD2 и ПЛИС EPM7128SLC84 (ф. Altera) DD3. В контроллере реализован PS/2 интерфейс и вся предварительная обработка информации, принимаемой от мышки и клавиатуры и передаваемой далее в ПЛИС. Прием и обработка скан-кодов клавиатуры происходит по прерываниям от линии CLK клавиатуры. Обработка заключается в перекодировке скан-кодов в соответствии с форматом данных Spectrum-клавиатуры. При этом в ОЗУ контроллера формируется вектор из 5 байтов (40 бит), таким образом, что каждый бит соответствует состоянию одной из клавиш ZX-клавиатуры (вспомним, что ZX-клавиатура представляет собой матрицу из 5x8 клавиш). Как нетрудно догадаться, такое представление данных ZX-клавиатуры дает исчерпывающую информацию о ее состоянии.

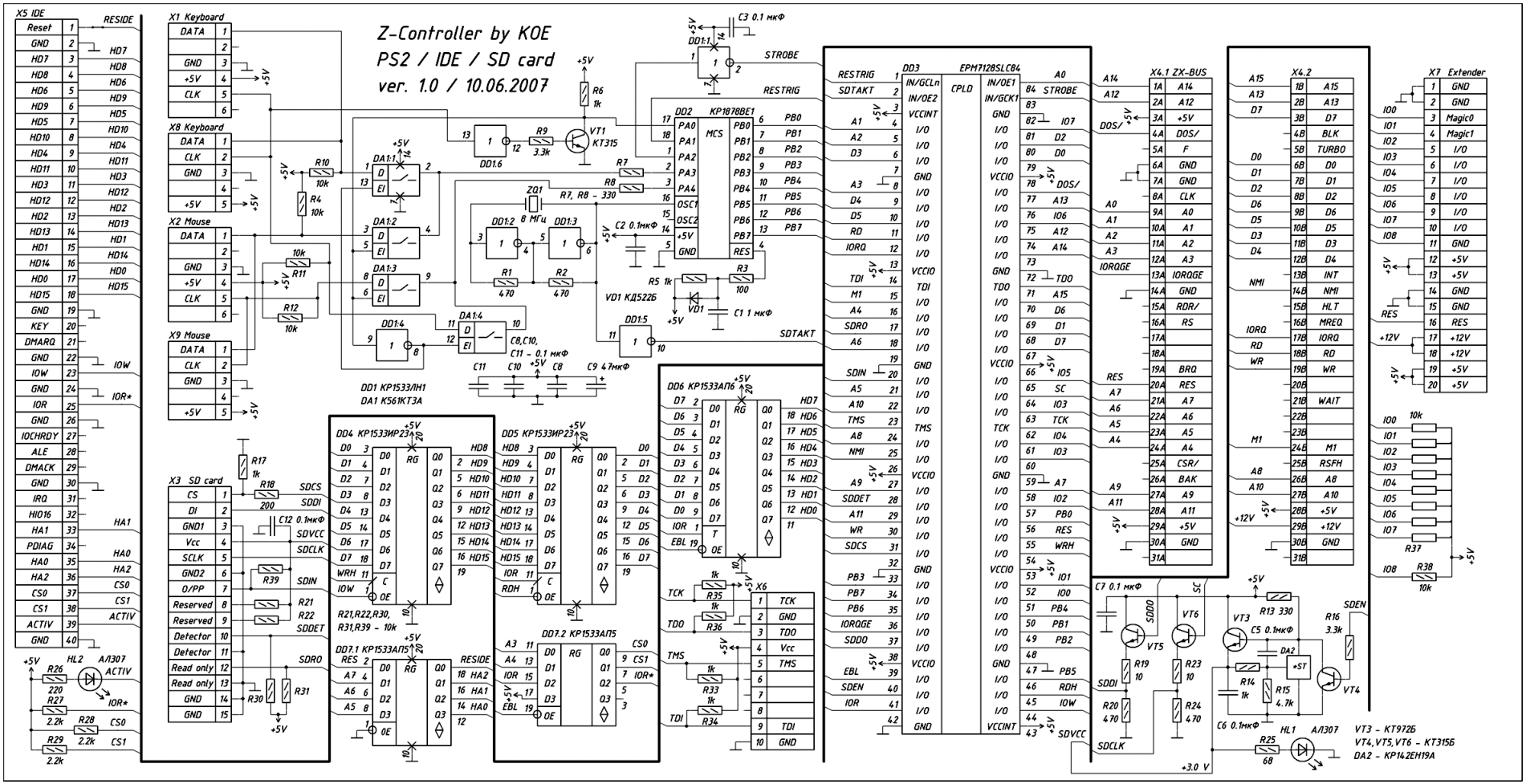

Рис.2. Схема электрическая принципиальная

Мышка опрашивается по прерываниям от встроенного в контроллер таймера. При работе с мышью используется так называемый “Remote mode”, в котором мышка передает данные о своем текущем состоянии после получения команды запроса статуса. После получения такой команды мышь возвращает код подтверждения (дабы дать понять хосту, что она находится в рабочем пока еще состоянии и адекватно воспринимает внешние воздействия) и 3 байта, соответствующие координатам X и Y указателя мышки и состоянию ее клавиш. Эти данные в контроллере также проходят перекодировку, в результате которой получаются 3 байта данных, записываемых непосредственно в порты ввода-вывода мышки. Эти 3 байта добавляются к вектору данных клавиатуры и полученный вектор длиной в 8 байтов передается в ПЛИС каждый раз при обработке прерывания от таймера микроконтроллером, либо при нажатии какой-либо клавиши. Для работы с мышкой со стороны Spectrum-а используются стандартные порты Kempston mouse.

Внутри ПЛИС реализованы порты ввода-вывода мышки, матрица клавиатуры, дешифратор портов IDE и SPI интерфейс, работа которого ниже будет описана.

Разъем X6 предназначен для подключения программатора ПЛИС. Незадействованные выводы ПЛИС подключены к разъему X7. Также к разъему подключены сигналы Magic0 и Magic1, предназначенные для соединения с соответствующими цепями в компьютере Pentagon-0124sl 2.2, для эмуляции замыкания кнопки ‘MAGIC’ клавишей F11.

При включении компьютера с установленным контроллером механическая клавиатура будет заблокирована, вместо нее можно использовать PS/2 клавиатуру.

Т.к. в последнее время развелось достаточно большое количество PS/2 мышек с различной разрешающей способностью, то в алгоритме предварительной обработки информации в микроконтроллере предусмотрена процедура масштабирования координат указателя мышки. Нажимая на PS/2 клавиатуре клавиши F2 и F3, можно менять масштабный коэффициент для координат (и как следствие, скорость перемещения по экрану указателя мышки). Для сохранения текущего значения масштабного коэффициента во flash-память микроконтроллера, достаточно нажать F1. При включении питания или после сброса контроллер автоматически восстанавливает сохраненное значения масштабного коэффициента.

IDE интерфейс в контроллере полностью (аппаратно и программно) совместим с контроллером IDE Вячеслава Скутина (Nemo). Т.к. ничего нового в этом смысле не сделано, не будем подробно разбирать его функционирование. Скажем лишь, что внутри ПЛИС находится дешифратор его портов ввода-вывода, а буферные микросхемы (DD4 … DD7), с помощью которых сделан обмен информацией между 16-разрядным IDE интерфейсом и 8-разрядной шиной данных Z80 последовательно по 8 бит, выведены за пределы ПЛИС из соображений экономии ее ресурсов.

Для подключения SD-карты в ZC реализован последовательный интерфейс передачи данных SPI. Программно со стороны Спектрума интерфейс доступен с помощью двух портов ввода-вывода: порта конфигурации 77h и порта данных 57h. Назначение битов в порте 77h следующее:

На запись:

bit 0 – питание SD-карты (0 – выключено, 1 - включено)
bit 1 – управление сигналом CS
bit 2..7 – не используются

На чтение:

bit 0 – если 0 – SD-карта установлена, 1 – SD-карта отсутствует
bit 1 - если 1 – то на карте включен режим Read only, если 0 – режим Read only не включен
bit 2..7 – не используются.

Порт данных 57h используется как на запись, так и на чтение для обмена данными по SPI-интерфейсу. Временная диаграмма передачи байта данных по SPI показана на рис.3.

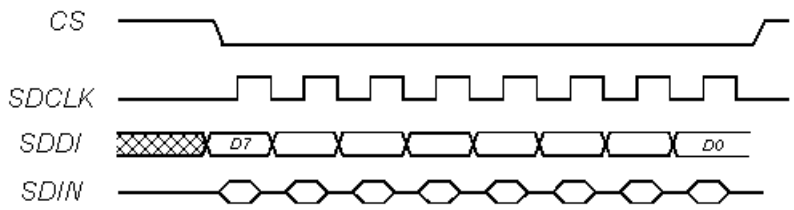

Рис.3. Временная диаграмма передачи байта данных по SPI в Z-Controller-е

Тактирование (сигнал SDCLK) осуществляется автоматически при записи какого-либо значения в порт 57h. При этом формируются 8 тактовых импульсов на выходе SDCLK, на выход SDDI поступают данные последовательно от старшего бита к младшему с каждым фронтом сигнала SDCLK. Период следования тактовых импульсов составляет 125 нс.

При чтении из порта 57h также автоматически производится тактирование. Буферный регистр порта 57h, используемый при чтении, заполняется данными со входа SDIN последовательно от старшего бита к младшему с каждым фронтом сигнала SDCLK.

На транзисторах VT5 и VT6 сделана схема согласования уровней между ПЛИС и SD-картой.

Немного о спецификации, SD и файловой системе.

Карты памяти SD (Secure Digital) является одной из разновидностей флэш-памяти, такие же как и обычные USB-Flash накопители, но основной ее особенностью является ставка на защиту записанной информации.

Разновидностей SD карт на сегодняшний день доступно несколько видов. Различаются они как по физическому размеру, так и по объему. Основным является стандартный размер, для карт размера Mini и Micro существуют переходники до стандартного размера.

Различий по объему всего два: до 2 Гб включительно карта называется стандартной и не несет на себе никаких дополнительных указаний кроме объема. Карты начиная с 4 Гб обязательно должны нести на себе гордую надпись SDHC (SD High Capacity) и Class 2 (4 или 6), цифра указывает скорость передачи данных в мегабайтах. Если на карте с объемом 4 Гб и более нет таких надписей, то карта просто неправильно маркирована.

Набор внутренних команд зависит от спецификации и режима работы. На сегодняшний день последняя версия спецификации имеет номер 2.0 и датирована 25 сентября 2006 года и существует на английском языке (возможно и на каких-то других языках), перевода на великий и могучий не обнаружено. Согласно этой спецификации объем карты до 2 Гигабайт включительно относится к стандартным по объему. Размер сектора можно менять программно от 1 до 512 байт, по умолчанию размер сектора равен 512 байтам. Карты объемом от 4 до 32 Гигабайт (ограничение этой версии спецификации на объем) имеют фиксированный размер сектора равный 512 байтам и не может быть изменен.

Карта может работать в одном из двух режимов: непосредственно SD режим и SPI. Для нас наиболее интересен режим SPI, который и реализован в Z-controller-е. Количество команд в SPI режиме значительно меньше, чем в режиме SD. На текущий момент написан и отлажен драйвер для работы с картами любого объема. Не проверена работа с картами калибра Mini и Micro ввиду их отсутствия, но скорее всего они будут работать.

Вообще спецификаций существует несколько. В свободном доступе была найдена только достаточно полная спецификация физического уровня с некоторыми удаленными главами по которой и был написан драйвер. Не была обнаружена спецификация файловой системы, которую весьма интересно было бы изучить. Дело в том, что после изучения описания файловой системы FAT можно по объему карты прикинуть, какая именно FAT используется на карте данного объема по умолчанию. Так вот карты до 2 Гб включительно без проблем вписываются в размер FAT16 (возможно отформатировать карту и в FAT32, защита на это никак не реагирует) и продаются отформатированными именно в FAT16. Достаточно много устройств использующих SD карт имеют ограничение на объем, чаще всего 0,5-1 Гб, возможно до 2 Гб. Раньше я этого ограничения не понимал, но не имея спецификации файловой системы могу предположить, что применение FAT32 предполагает наличие большего объема встроенной памяти в устройстве для работы с этой файловой системой. А устройств рассчитанных на применение FAT32 на данный момент или нет вообще или их достаточно

мало, хотя это уже возможно и изменилось. А карты объемом 4 Гб и выше могут нести на себе только FAT32. Это подтвердилось приобретением карты SDHC объемом 4 Гб. Но удивило не наличие на этой карте FAT32, а размер кластера. Фактически, размер кластера на свежеприобретенной карте объемом 4 Гб при формате FAT32, равен 64 сектора (32 Кб), в то время как на свежеприобретенной карте объемом 1 Гб при формате FAT16 размер кластера равен 32 сектора (16 Кб). При таких размерах кластера на FAT32 потери свободного места будут больше, чем на FAT16. Что и как прописано в спецификации файловой системы для SD карт остается только гадать и не понятно, почему сделан такой огромный размер кластера. Чтобы уменьшить потери свободного места карты объемом 4 Гб и выше придется переформатировать с уменьшением размера кластера. После форматирования через кардридер карта объемом 4 Гб из FAT32 опять в FAT32 размер кластера уменьшился до 8 секторов (4 Кб).

Кстати, с разрядностью FATа связан один нюанс. Если между FAT12 и FAT16, кроме разрядности и некоторых изменений в описателе разницы особой нет, то у FAT32 структура описателя несколько сложней. Самое главное, что из-за размера FAT таблиц в структуру описателя введен добавочный сектор, в который заносится информация о количестве свободных кластеров и номере кластере, с которого драйвер должен искать свободные кластера для записи. А с учетом того, что при записи на FAT этот сектор должен постоянно обновляться, а количество записей на карту ограниченно количеством 200-300 тысяч, то срок жизни этого сектора будет значительно меньше срока жизни самой карты при использовании на том же пц. И не имеет особого значения, что у этого сектора есть копия. Копия-то как раз и есть, но всего лишь копия и признаков ее использования той же виндой не заметно. А работа с SD картой через кардридер выявила еще и то, что тот же XP с FAT-таблицами работает с ошибкой. А на Спектруме, если карта достаточно сильно забита файлами, приходится свободный кластер искать с самого начала таблицы, что может занять довольно большое время (для карт объемом 1 Гб, форматированной как FAT32 и заполненной чуть больше половины время поиска в турбо-режиме до 7 секунд). Если карта будет использоваться для записи только Спектруме, то эта ошибка особого значения иметь не будет.

Описание драйвера

При написании драйвера использовалось описание инициализации от создателя Хард-тапер и исходники с сайта www.zxbada.bbk.org для работы с MMC картой. Драйвер писался из расчета поддержки только SD карт и не поддерживает MMC карты по причине отсутствия как самих карт для проверки, так и спецификаций на них в свободном доступе. При попытке использования MMC карт с большой долей вероятности драйвер будет банально зависать. О причинах - в тексте драйвера.

Текст драйвера снят с рабочего исходника и является полностью рабочим. В этом виде драйвер занимает около 530 байт. Данный драйвер поддерживает карты любого объема в пределах текущей спецификации, определения типа карты делается во время исполнения команды чтения/записи.

Размер сектора задается командой с кодом 16 и возможен только для стандартных карт. По умолчанию размер сектора равен 512 байт и изменение его в драйвере не предусмотрено в связи с поддержкой карт калибра SDHC (4 Гигабайт и более), у которых изменение размера сектора не возможно. У многосекторных команд чтения/записи нет счетчика количества секторов, остановка чтения/записи осуществляется соответствующими командами остановки.

Команд чтения/записи есть две разновидности: для работы с одним сектором и многосекторная. Зачем сделано это разделение понять трудно, для односекторной работы можно использовать и многосекторную команду, я это проверил, но для себя оставил как есть.

На всех SD картах есть область между MBR и началом раздела, которая по всей видимости используется при шифровании содержимого карты. Размер этой области зависит от объема карты и производителя.

Размер любой команды для карты равен 6 байтам (48 бит). Вся команда передается в саму карту от старшего бита до младшего. Формат команды:

Номер бита:	47	46	45-40	39-8	7-1	0
Размер, бит:	1	1	6	32	7	1
Значение:	0	1	X	X	X	1
Описание:	Стартовый бит	Направление передачи	Индекс команды	Аргумент	CRC7	Стоповый бит

После подачи команды первый считанный байт (не равный #FF) является ответом карты и если не равен нулю (команды будет выполнена), то содержит биты ошибки. В драйвере код ошибки чаще всего игнорируется. По спецификации карта выдает несколько разновидностей ответов различающихся длиной, в драйвере используется только первый байт ответа.

Коды ошибок по битам. Возможно я где-то не так перевел, поэтому привожу оригинальное описание битов ошибок:

7-всегда 0
6-parameter error (ошибочный параметр)
5-address error (неправильный адрес блока)
4-erase sequence error
3-com crc error (ошибка CRC команды (при отключенном CRC не должна появляться)
2-illegal command (неизвестная команда)
1-erase reset
0-in idle state (карта находится в режиме инициализации и недоступна)

Неправильный адрес блока появляется только при превышении максимального номера сектора. Неизвестная команда скорее всего будет появляться в ответ на команды которые не поддерживаются. В режиме SPI многие команды SD-режима недоступны.

;Пример использования драйвера:

```
            CALL  COM__SD
            DB 1
```

;включение и инициализация карты памяти, на выходе A - смотри коды возвращаемых ;ошибок

```
            LD HL, АДРЕС ЗАГРУЗКИ
            LD BC, СТАРШИЕ 16 БИТ НОМЕРА СЕКТОРА
            LD DE, МЛАДШИЕ 16 БИТ НОМЕРА СЕКТОРА
            LD A, КОЛИЧЕСТВО БЛОКОВ
            CALL  COM__SD
            DB 3
```

;чтение заданного количества секторов по заданному адресу начиная с заданного сектора
;на выходе не забываем проверять код ошибки в A

```
            CALL  COM__SD
            DB 1
```

;выключаем питание карты

```
;Драйвер SD карты
;LAST UPDATE 19.02.2008 savelij

;Первые две функции входных параметров не имеют.
;Входные параметры общие:
;HL-адрес загрузки в память/запись из памяти
;BCDE-32-х битный номер сектора
;A-количество секторов (сектор=512 байт), только для многосекторного
;чтения/записи
;Выходные параметры:
;HL-адрес следующего байта после окончания чтения/записи
;BCDE-не изменяется
;A-смотри коды ошибок, альтернативный A портится

;Ошибки выдаваемые на выходе драйвера:
;A=0-инициализация прошла успешно или команда выполнена успешно
;A=1-карта отсутствует или не ответила
;A=2-карта защищена от записи
;A=3-попытка записи в сектор 0 карты

;адреса портов контроллера
P_DATA  EQU #57  ;порт данных
P_CONF  EQU #77  ;порт конфигурации
```

```
;некоторые коды команд SD карт используемые в драйвере
CMD_12  EQU #41 ;STOP_TRANSMISSION остановка чтения
CMD_17  EQU #4A ;READ_SINGLE_BLOCK чтение одиночного блока
CMD_18  EQU #4C ;READ_MULTIPLE_BLOCK чтение нескольких блоков
CMD_24  EQU #58 ;WRITE_BLOCK запись одиночного блока
CMD_25  EQU #59 ;WRITE_MULTIPLE_BLOCK запись нескольких блоков
CMD_55  EQU #77 ;APP_CMD последующая команда дополнительная
CMD_59  EQU #7A ;CRC_ON_OFF команда принудительного отключения CRC16
ACMD_41 EQU #69 ;SD_SEND_OP_COND дополнительная команда сброса

;ОБЩАЯ ТОЧКА ВХОДА ДЛЯ РАБОТЫ С SD
;код команды должен быть после команды вызова драйвера, без
;комментариев
COM__SD EX (SP),HL
        EX AF,AF
        LD A,(HL)
        INC HL
        EX (SP),HL
        PUSH HL
        PUSH DE
        LD L,A
        LD H,0
        ADD HL,HL
        LD DE,TABL_SD
        ADD HL,DE
        LD E,(HL)
        INC HL
        LD D,(HL)
        EX DE,HL
        POP DE
        EX (SP),HL
        EX AF,AF
        RET
                    ;коды и описания функций
TABL_SD DW SD_INIT  ;0-инициализация карты
        DW SD__OFF  ;1-отключение питания карты
        DW RDSINGL  ;2-чтение одного сектора
        DW RDMULTI  ;3-чтение нескольких секторов
        DW WRSINGL  ;4-запись одного сектора
        DW WRMULTI  ;5-запись нескольких секторов

;подпрограмма инициализации
SD_INIT CALL CS_HIGH;включаем питание карты при не выбранной карте
        LD BC,P_DATA;сигнал выбора карты равен 1
        LD DE,#20FF ;записываем согласно спецификации в порт
        OUT (C),E   ;много единиц, главное отличие: количество
        DEC D       ;единиц несколько увеличено и больше, чем
        JR NZ,$-3   ;требуется по спецификации
        LD A,30     ;будем переводить карту в режим SPI до 30 раз
        EX AF,AF
ZAW001  LD HL,CMD00 ;даем команду сброса, эта команда после
        CALL OUTCOM ;включения питания переводит карту в режим SPI
        CALL IN_OOUT;читаем ответ
        EX AF,AF
        DEC A
        JR Z,ZAW003 ;если карта 30 раз не ответила, то карты нет
        EX AF,AF
        DEC A
        JR NZ,ZAW001
        LD HL,CMD08 ;запрос на поддерживаемые напряжения
```

```
        CALL OUTCOM ;команда поддерживается начиная со спецификации
        CALL IN_OOUT;версии 2.0 и только SDHC картами
        IN H,(C)    ;в A=код ответа карты
        IN H,(C)    ;считываем 4 байта длинного ответа
        IN H,(C)    ;но не используем
        IN H,(C)
        LD HL,0     ;HL=аргумент для команды инициализации
        BIT 2,A     ;если бит 2 установлен, то карты стандартная
;здесь содержимое A можно передать для подпрограммы SECM200
;смотри саму подпрограмму SECM200
        JR NZ,ZAW006
        LD H,#40    ;если сброшен, то карта SDHC
ZAW006  LD A,CMD_55 ;запускаем процесс внутренней инициализации
        CALL OUT_COM;для карт MMC здесь должна быть другая команда
        CALL IN_OOUT;соответственно наличие в слоте MMC-карты
        LD A,ACMD_41;вызовет зависание драйвера, от применение
        OUT (C),A   ;общей команды запуска инициализации я отказался
        OUT (C),H   ;бит 6 установлен для инициализации SDHC карты
        OUT (C),L   ;для стандартной сброшен
        OUT (C),L
        OUT (C),L
        LD A,#FF
        OUT (C),A
        CALL IN_OOUT;ждем перевода карты в режим готовности
        AND A       ;время ожидания примерно 1 секунда
        JR NZ,ZAW006
ZAW004  LD A,CMD_59 ;принудительно отключаем CRC16
        CALL OUT_COM;
        CALL IN_OOUT
        AND A
        JR NZ,ZAW004
ZAW005  LD HL,CMD16 ;принудительно задаем размер блока 512 байт
        CALL OUTCOM
        CALL IN_OOUT
        AND A
        JR NZ,ZAW005

;включение питания карты при снятом сигнале выбора карты
CS_HIGH PUSH DE
        PUSH BC
        LD E,3
        LD BC,P_CONF
        OUT (C),E   ;включаем питание, снимаем выбор карты
        LD E,0
        LD C,P_DATA ;обнуляем порт данных
        OUT (C),E  ;обнуление порта можно не делать, просто последний
        POP BC     ;записанный бит всегда 1, а при сбросе через вывод
        POP DE     ;данных карты напряжения попадает на вывод питания
        RET        ;карты и светодиод на питании подсвечивается

;возврат по не ответу карты с кодом ошибки 1
ZAW003  CALL SD__OFF;карта не найдена, выключаем питание
        LD A,1
        RET

;выключение питания карты
SD__OFF XOR A
        OUT (P_CONF),A
        OUT (P_DATA),A
        RET
```

```
;выбираем карту сигналом 0
CS__LOW PUSH DE
        PUSH BC
        LD E,1
        LD BC,P_CONF
        OUT (C),E
        POP BC
        POP DE
        RET

;запись в карту команды с неизменяемым параметром из памяти
OUTCOM  CALL CS__LOW
        PUSH BC
        LD BC,P_DATA
        OUTI        ;передаем 6 байт команды из памяти
        OUTI        ;адрес команды в HL
        OUTI
        OUTI
        OUTI
        OUTI
        POP BC
        RET

;запись в карту команды с нулевыми аргументами
;A-код команды, аргумент команды равен 0
OUT_COM PUSH BC
        CALL CS__LOW
        LD BC,P_DATA
        OUT (C),A
        XOR A
        OUT (C),A   ;пишем 4 нулевых байта аргумента
        OUT (C),A
        OUT (C),A
        OUT (C),A
        DEC A
        OUT (C),A   ;пишем пустой CRC7 и стоповый бит
        POP BC
        RET

;запись команды чтения/записи с номером сектора
;для карт стандартного размера при изменяемом размере сектора
;номер сектора нужно умножать на его размер, для карт SDHC размер
;сектора не требует умножения
SECM200 PUSH HL
        PUSH DE
        PUSH BC     ;BCDE=номер сектора
        PUSH AF
        PUSH BC
        LD HL,CMD08 ;сюда можно передать код ответа из подпрограммы
        CALL OUTCOM ;инициализации и еще раз проверив бит 2
        CALL IN_OOUT;принять решение о типе карты и дальнейших
        LD BC,P_DATA;действиях
        LD H,A      ;сохраняем код ответа
        IN A,(C)    ;пропускаем неиспользуемые байты
        INC A
        JR NZ,$-3
        BIT 2,H     ;проверяем на ошибку
        POP HL      ;теперь HLDE=номер сектора
        JR Z,SECN200;если ошибки нет, то умножение номера сектора
        EX DE,HL    ;на размер сектора не требуется
        ADD HL,HL
```

```
        EX DE,HL
        ADC HL,HL
        LD H,L
        LD L,D
        LD D,E
        LD E,0      ;сейчас HLDE=номер сектора*512
SECN200 POP AF
        CALL CS__LOW
        LD BC,P_DATA
        OUT (C),A   ;записываем код команды
        OUT (C),H   ;записываем 4 байта аргумента
        OUT (C),L   ;аргумент-HLDE=номер сектора
        OUT (C),D   ;пишем от старшего байта
        OUT (C),E   ;до младшего
        LD A,#FF
        OUT (C),A   ;пишем замену CRC7 и стоповый бит
        POP BC
        POP DE
        POP HL
        RET

;чтение ответа карты до 16 раз, если ответ не #FF-немедленный выход
IN_OOUT PUSH BC
        PUSH DE
        LD DE,#10FF
        LD BC,P_DATA
IN_WAIT IN A,(C)
        CP E
        JR NZ,IN_EXIT
IN_NEXT DEC D
        JR NZ,IN_WAIT
IN_EXIT POP DE
        POP BC
        RET

CMD00   DB #40,#00,#00,#00,#00,#95
;GO_IDLE_STATE команда сброса и перевода карты в SPI режим после
;включения питания
CMD08   DB #48,#00,#00,#01,#AA,#87
;SEND_IF_COND запрос поддерживаемых напряжений
CMD16   DB #50,#00,#00,#02,#00,#FF
;SET_BLOCKEN команда изменения размера блока на размер 512 байт

;читаем один сектор из карты в память
RD_SECT PUSH BC
        PUSH DE
        LD BC,P_DATA
        INIR        ;читаем первые 256 байт
        INIR        ;читаем следующие 256 байт
        IN A,(C)    ;здесь считываем CRC16 выдаваемые
        IN A,(C)    ;картой, но не используем
        POP DE      ;замена INIR на кучу INI мало что дает
        POP BC      ;хотя я могу и ошибаться
        RET

;записываем один сектор из памяти в карту
WR_SECT PUSH BC
        PUSH DE
        LD BC,P_DATA
        OUT (C),A   ;пишем код команды
        OTIR        ;пишем первые 256 байт
```

```
        OTIR        ;пишем следующие 256 байт
        LD A,#FF
        OUT (C),A   ;записываем замену CRC16
        OUT (C),A   ;равное #FFFF
        POP DE
        POP BC
        RET

;многосекторное чтение
RDMULTI EX AF,AF    ;прячем счетчик секторов
        LD A,CMD_18 ;даем команду многосекторного чтения
        CALL SECM200
        EX AF,AF
RDMULT1 EX AF,AF
        CALL IN_OOUT;пропускаем код ошибки
        CP #FE      ;и ждем токен готовности #FE для начала чтения
        JR NZ,$-5
        CALL RD_SECT;читаем сектор
        EX AF,AF
        DEC A
        JR NZ,RDMULT1;читаем сектора пока не обнулится счетчик
        LD A,CMD_12 ;по окончании чтения даем команду карте «СТОП»
        CALL OUT_COM;команда мультичтения не имеет счетчика и
        CALL IN_OOUT;должна останавливаться здесь командой 12
        INC A
        JR NZ,$-4   ;ждем освобождения карты
        JP CS_HIGH  ;снимаем выбор с карты и выходим с кодом 0

;чтение одного блока
RDSINGL LD A,CMD_17 ;даем команду чтения одного сектора
        CALL SECM200
        CALL IN_OOUT;пропускаем код ошибки
        CP #FE      ;и ждем токен готовности #FE для начала чтения
        JR NZ,$-5
        CALL RD_SECT;читаем сектор
        CALL IN_OOUT
        INC A
        JR NZ,$-4   ;ждем освобождения карты
        JP CS_HIGH  ;снимаем выбор карты и выходим с кодом 0

;запись одиночного блока
WRSINGL XOR A
        IN A,(P_CONF);проверяем защиту от записи
        AND 2       ;если защита включена выходим с кодом 2
        RET NZ
        LD A,B      ;проверяем на попытку записи в нулевой
        OR C        ;сектор, сделано на всякий пожарный
        OR D        ;я пока эксперментировал одной карте MBR затер
        OR E
        LD A,3      ;если сектор нулевой, выходим с кодом 3
        RET Z
        LD A,CMD_24 ;даем команду записи одного сектора
        CALL SECM200
        CALL IN_OOUT;пропускаем код ошибки
        INC A
        JR NZ,$-4   ;и ждем освобождения карты
        LD A,#FE    ;пишем стартовый токен, сам блок и пустое CRC16
        CALL WR_SECT
        CALL IN_OOUT
        INC A
        JR NZ,$-4   ;ждем освобождение карты
```

```
        JP CS_HIGH  ;снимаем выбор карты и выходим с кодом 0

;многосекторная запись
WRMULTI EX AF,AF    ;сохраняем счетчик блоков
        XOR A
        IN A,(P_CONF);как и в случае с записью одного сектора
        AND 2       ;проверяем защиту от записи
        RET NZ
        LD A,B      ;и попытку записи в MBR
        OR C
        OR D
        OR E
        LD A,3
        RET Z
        LD A,CMD_25 ;даем команду мультисекторной записи
        CALL SECM200
        CALL IN_OOUT
        INC A
        JR NZ,$-4   ;ждем освобождения карты
        EX AF,AF
WRMULT1 EX AF,AF
        LD A,#FC    ;пишем стартовый токен, сам блок и пустое CRC16
        CALL WR_SECT
        CALL IN_OOUT;пропускаем код ошибки
        INC A
        JR NZ,$-4   ;и ждем освобождения карты
        EX AF,AF
        DEC A
        JR NZ,WRMULT1;продолжаем пока счетчик не обнулится
        LD C,P_DATA
        LD A,#FD
        OUT (C),A   ;даем команду остановки записи
        CALL IN_OOUT
        INC A
        JR NZ,$-4   ;ждем освобождения карты
        JP CS_HIGH  ;снимаем выбор карты и выходим с кодом 0
```