SD Card and SDOS for 8-bit Computers

Author: PVV (Vitaly V. Popov, Rostov-on-Don)

Source: https://zx-pk.ru/threads/29892-sd-karta-i-sdos-dlya-8i-bitnykh-pk.html
Author: https://www.qrz.ru/db/RX6LEJ

http://www.nedopc.org/forum/viewtopic.php?f=35&t=9834&hilit=sdos
http://www.nedopc.org/forum/download/file.php?id=6181 (version 8.9J, Jupiter ACE, latest found)

The most recent version found: v8.C from 11.03.2020, 23:15
https://zx-pk.ru/attachment.php?attachmentid=71773&d=1583957695

I have a desire to organize and compile all the information about the SDOS system in one place. Honestly, I did not anticipate that SDOS would grow to its current scale, and now I sometimes face difficulty finding information about a specific build.

What is SDOS?

SDOS is a compact console shell that allows reading and executing files from an SD card formatted in FAT16. It can also write data back to the card from the computer's RAM. This makes it very easy to transfer files between 'big' PCs and our retrocomputers. The codebase of SDOS is based on the code from the respected **b2m** (http://bashkiria-2m.narod.ru), which I framed with additional functionality and capabilities. The system code is designed to be easily adaptable to any PC and any hardware interface and storage medium (it doesn't necessarily have to be an SD card; it can be an HDD).

SDOS properties:

- The code itself occupies from 2 to 2.5 KB (can be located in both ROM and RAM), and an additional 2 KB of RAM is required to work with FAT16.
- Has been compiled and tested at least in the emu emulator, and in some cases, in real life on the following PCs: Specialist std (real) and MX2 (real), RK-86 (real), Apogee, Partner-01, Galaksija (real), Orion (real), TRS-80 (real), YUT-88.
- Supports three basic SD hardware interfaces: HWM_PVV, msx, n8vem, and their derived variants.
- Supports SD and SDHC cards, i.e., cards larger than 4 GB.

Supports the following basic directives:

- **`CD`** `DirectoryName`
  change to the directory with the specified name;

- **`DIR`**
  display the list of files and directories;

- `FileName.RKX`
  (**RKS** for std, **RKR** for RK-86, **GTP** for Galaksija, **RKO** for Orion, **CAS** for TRS-80) - execute the file; the extension can be omitted, and autocompletion will occur;

- **`R`** `FileName.Extension, Address, Size`
  read the file without executing it, starting from the specified memory address and reading the specified number of bytes.

Example: R TEST.BIN,0ACD,5FE0 - reads the TEST.BIN file into memory starting from address 0x0ACD and ending at address 0x0ACD+0x5FE0=0x6AAD. There is no check for the actual length of the file and the requested read size, so you can request to read more than the file's size, and the behavior is undefined.

- **`W`** `FileName.Extension, Address, Size`
  write data from memory to the file, starting from the specified address in memory and writing the specified number of bytes.

  Example: W TEST.BIN,0ACD,5FE0 - writes to the TEST.BIN file from memory starting from address 0x0ACD and ending at address 0x0ACD+0x5FE0=0x6AAD. There is no check for the actual length of the file and the requested write size, so you can request to write more than the file's size, and the behavior is undefined. It is not possible to write more than the existing file size. If you write less data than the file size, the file size remains unchanged.

- **`X`**
  switch to the monitor from which SDOS was launched;

- **`I`**
  initiate card reinitialization.

Only for Galaksija:

- **`WB`** `FileName.Extension`
  write BASIC program to the SD file in GTP format.

Only for SpecialistMX:

- **`L`** `FileName.Extension`
  read the file in RKX format from SD to MX RAM disk; example: L TEST.BIN - reads the TEST.BIN file into memory starting from address 0x0000 and ending at its length address, creates a file TEST.BIN on the RAM disk with a start address and size taken from the first 4 bytes of the file on the card, assuming that the file name in the header was absent (0xE6 in the 5th byte). Or: L TEST.BIN - reads the TEST.BIN file into memory starting from address 0x0000 and ending at its length address, creates a file TEST_MX.HEX on the RAM disk with the start address and size read from the first 4 bytes of the file on the card, and the file name in the header is TEST_MX.HEX.

- **`S`** `FileNameOnRAMDisk.Extension, FileNameOnSD.Extension`
  write the file data from RAM disk to SD in RKX format; example: S TEST.BIN - reads the TEST.BIN file from RAM disk into memory starting from address 0x0000 and ending at its length address, then writes it to the SD card in the existing file with the same name TEST.BIN, the size remains unchanged, RKX format! Or: S TEST.BIN,TEST_MX.HEX - reads the TEST_MX.HEX file into memory starting from address 0x0000 and ending at its length address, then writes it to the SD card in the existing file with the name TEST.BIN.

Only for Orion:

- **`L`** `FileName.Extension`
  read the file in RKO format from SD to RAM disk; example: L TEST$.RKO - reads the

TEST$.RKO file into memory starting from address 0x0000 and ending at its length address, creates a file TEST$ on the RAM disk with the name, start address, and size taken from the RKO file header on the card.

- `S FileNameOnRAMDisk, FileNameOnSD.Extension`
  write the file data from RAM disk to SD in RKO format; example: S TEST$,TEST1$.BIN - reads the TEST$ file from RAM disk into memory starting from address 0x0000 and ending at its length address, then writes it to the SD card in the existing file with the name TEST1$.BIN, the size remains unchanged, RKO format!

When listing the directory with DIR, the name, file extension, and size of the file are printed (so that you can use R and W directives), and for directories, DIR is written in the file size field.

When starting a file, the start and end addresses to which the file will be read from the card are written.

Pressing any key (or space) during DIR output pauses the file list output.

For SpecialistMX2, the ability to run not only RKX files but also RKS is implemented. To do this, in the same directory where the RKS file is located, you need to place the M2_C000.MON monitor file (taken from the Vinxru project). In addition, there is the possibility to load arbitrary monitors.

And now a lot of links, mentioning SDOS and SD interface.

It all started with this thread about Specialist with the HWM interface, and back then, I thought it would end there, but it didn't. The first attempt to organize information about SDOS within the framework of Specialist.

Version for RK-86 with HWM_PVV interface, then with RK86_WW55_SD_HWM_PVV interface, then Apogee with the original card connection option, an analog of the SD_n8vem interface. In these variants, the SD card is connected together with the ROM disk, without excluding it!
Partner-01.

Version for Galaksija with HWM_PVV interface and with SD_n8vem interface.
Practical implementation in this thread.

YUT-88 with SD_n8vem interface.
Orion.
TRS-80.

Interface schematics:

HWM_PVV on IR8 and IR9 or on ports VV55.
HWM_PVV on IR24 or here or here.
HWM_PVV on IR13.

SD_n8vem or here or here or here.

I haven't tested SD_msx in real life, only in emulation, but initially, b2m's code was implemented to work specifically with this interface, and it was used in real life on PLD.

I want to add one more point. According to the specification, when initializing the SD card, you

need to use an SD_CLK clock frequency not exceeding 400 kHz, and after initialization, switch to a higher frequency. But as practice has shown, modern cards easily initialize and work at a frequency of 2-4 MHz, allowing you to dispense with the clock speed switch. Speed switching is only supported in the HWM PLD interface and in the first version of the HWM_PVV interface on discrete components.

…

The source code for SDOS is open, and it can be downloaded from the link above.

SDOS is written in i8080 mnemonics (actually i8085) and runs on Z80 without any problems.

Each time I added any functionality or new platforms (PCs), I increased version and subversion numbers.

The current version is 8.9, but practically, it's 8.6, since 8.7 - 8.9 are just builds for different platforms with subvariants of the SD interface. At this stage, I realized that it shouldn't be like this; the end-user should build SDOS according to their requirements, and it shouldn't affect the system version.

Accordingly, I want to explain how SDOS is assembled, configured, or ported to another platform.

First, you need to determine the placement in the address space of the target platform. You need to decide where both the code of SDOS itself and its service area 2048-256=1792 bytes will be located, and at which addresses. You need to choose addresses that are not used by most programs for the best part. Here, you need to choose the optimal compromise. If the target platform has an external ROM disk, it is logical to place SDOS there and rewrite it into RAM by standard means during operation. (When porting to a new platform, you can choose almost any available addresses at the initial stage, and only at the final stage, determine the optimal memory placement). If there is free space in the address space of the target platform where you can place ROM (for example, Specialist and Galaxia), then SDOS is placed in such an area, and it works immediately from ROM. You just need to choose an available area for the service region.

Second, it overlaps with the first, which is choosing addresses for the ports of the SD interface. Two addresses are needed, which can be either addresses in the memory space or addresses in IO ports. For platforms with an external ROM disk, the choice may be simplified. There are two connection options in the links above. Otherwise, you need to search for options and compromises again.

Third, choose the SD interface variant. HWM_PVV is the fastest, and, in addition, its software support is the most compact, which is quite noticeable against the background of the compact SDOS (remember ~2KB total!). 7 bodies of the 555 series. MSX is slower, and software support adds +150 bytes to the code, and in terms of the number of cases, it is approximately like the previous option. Therefore, I don't particularly consider it. N8VEM is the slowest, +250 bytes to the code, but it is very simple to implement, and therefore, it has the right to be considered. For our retro PCs, even the speed of N8VEM is not so bad because most files have a small size, which is not critical and still better than just a cassette recorder. Any subvariants of these interfaces, for example, connected via VV5, will be slightly slower and slightly inflate the program code on the whole.

At this stage, for existing platform variants, it is enough to correctly define the necessary capabilities in the defines and run the assembly with a bat file.

If you need to port SDOS to a new platform, then some research is needed here. In the defs.inc file, you need to make a section with a new platform, in which you need to define such functions (an example from RK-86):

```assembly
GETC EQU 0F803H ; Wait for a character from the keyboard
PUTC EQU 0F809H ; Output one character to the screen
PRINT EQU 0F818H ; Output a string to the screen
PRHEX EQU 0F815H ; Output a number in HEX format
IfKeyPress EQU 0F81BH ; Non-hanging keyboard input check
```

You can see the input and output parameters of these functions, for example, here. The main ones are the first two, as PRINT is made from PUTC, PRHEX as an option, is already in the SDOS code and uses PUTC, and IfKeyPress can be commented out and not used (there will be no function to pause the file list output by pressing any key during DIR, forgot to mention this feature in the first message). GETC and PUTC are somehow implemented on the target platform, so you need to adjust their call format to the required SDOS (this was done when porting to Galaxia and TRS-80).

Here are the defines in the defs.inc file and what they mean (the presence of a semicolon ';' at the beginning of the line indicates the exclusion of this define, and its absence indicates inclusion):

```assembly
#define SDOS_VER "SDOS_V8.9N" ; In v8.9, moved the greeting text of SDOS to this define

#define SD_DBG_PRINT0 ; Also in v8.9, split one define SD_DBG_PRINT into three for
#define SD_DBG_PRINT1 ; more flexible adjustment of the SDOS size, which is crucial when building compact code
#define SD_DBG_PRINT2 ; for placement in 2KB ROM (all these prints 'consume' up to 100 bytes of code).

;#define SD_HWM_PVV - this group of 8 defines determines the type of SD interface for assembly, corresponding to HWM_PVV
;#define SD_msx       - MSX
;#define SD_n8vem    - N8VEM
;#define RK86_WW55_SD_HWM_PVV     - this is the HWM_PVV variant connected to VV55 ports, tested on a real RK-86, hence the name
;#define RK86_WW55_SD_n8vem ; APOGEE and RK86 memmap - this is the N8VEM variant connected to VV55 ports, where VV55 is placed in the address space, not in the ports, assembled but not yet tested on RK86
#define STD_WW55_SD_n8vem ; Specialist - this is a complete copy (!?!) of the RK86_WW55_SD_n8vem variant, assembled but not tested on a real system, for the Specialist (Std) - in the next version, I will combine it into one define WW55_ADDR_SD_n8vem
;#define UT88_WW55_SD_n8vem ; ports IN - OUT - this is the N8VEM variant connected to VV55 ports, where VV55 is placed in IO ports, assembled but not yet tested on UT-88 - in the next version, I will rename the define to WW55_IO_SD_n8vem
;#define GAL_AY_SD_n8vem          - this is the N8VEM variant connected to AY3-8910(YM2149F) ports, where AY is placed in IO ports, tested on a real Galaxia - in the next version, I will rename the define to AY_IO_SD_n8vem

;#define FAT12_ON   - enabling FAT12 support on the card, relevant for cards or partitions less than 32MB, usually turned off for code size savings
```

#define FAT16_ON - enabling FAT16 support on the card, the main one

#define RWR - enabling write support to the card, takes up 200-300 bytes

;#define UT88- this group of 8 defines determines the type of platform for which SDOS is assembled. UT-88
;#define APOGEE - Apogee
;#define RK86 - RK-86
#define STD - Specialist
;#define MX2 - SpecialistMX2 - relevant for the original SD_HWM_PVV, where the bit signaling 'BUSY' of the SPI state machine is involved (in all 'my' variations of this interface, I dropped this bit because knowing the CPU speed with the SD interface, you can adjust the required delay with NOPs, literally from 1 to 3 pieces)
;#define GAL - Galaxia
;#define ORION - Orion
;#define TRS80 - TRS80
```

Here's a translation of the section about platform-specific defines and the list of source code files:

```assembly
;================================================ =========
#ifdef RK86
GETC EQU 0F803H - wait for a character from the keyboard
PUTC EQU 0F809H - output one character to the screen
PRINT EQU 0F818H - output a string to the screen
PRHEX EQU 0F815H - output a number in HEX format
IfKeyPress EQU 0F81BH - keyboard input check, non-blocking
SD_DATA_PORT EQU 0d000H - address where the SD interface and its data port are located
SD_CONF_PORT EQU SD_DATA_PORT+1 - address where the SD interface and its configuration port are located
SD_DATA_PORT_WW55 EQU 0a000H - address where the SD interface and its VV55 port are located for SD variants with VV55
#define SD_ROM - relevant only for RK-86; if not selected, a jmp F800h is placed at address F000
#ifdef SD_ROM
START_ADDR EQU 06000H - starting address for placing the SDOS code (read from RK's ROMdisk)
#else
START_ADDR EQU 0f000H - starting address for placing the SDOS code if it is stored in ROM
#endif
BUF EQU 06900h - starting address for placing the service area (buffers for working with FAT), takes up 2048-256=1792 bytes
#endif
;================================================ =========
```

The source codes of SDOS are in several files:

- defs.inc - contains all the defines described earlier

- dos_rk.asm - the main file, in which the implementation of the entire interface for interacting with SDOS and the user is located
```

There are six files for each of the supported platforms, implementing the file reading and execution function, plus specific features for a particular platform (working with RAMdisk in SpecialistMX2 or Orion, writing basic files in Galaxia).

When building, the file of the platform defined in the defs.inc file is used. The file is included in the dos_rk.asm file.

When adding a new platform, you will need to add such a file.
- dos_gal_RUNF.inc
- dos_mx_RUNF.inc
- dos_orion_RUNF.inc
- dos_rk_RUNF.inc
- dos_std_RUNF.inc
- dos_trs_RUNF.inc

- sd_proc.inc - contains all functions for working with the SD card itself

- fs_proc.inc - contains all functions for working with the FAT12 and FAT16 file systems.

In addition to the above files, the SDOS archive also includes:

- asm_std.cmd - a text BAT file that describes the SDOS build options; you need to run it for the build after configuring all the defines.

- ReadMe.txt - a text file with a very brief description of SDOS capabilities

- TASM.EXE - this is the assembler used for building

- TASM85.TAB - this is an assembler auxiliary file

The entire build process for the user consists of configuring the defines in the defs.inc file and executing asm_std.cmd. The result will be the required SDOS.BIN file.

Here is a translation of the configuration details for the emulator:

There is always a desire to check what has been achieved, and in this case, the emulator "emu" becomes a tremendous help. This emulator supports all three types of SD interfaces and their variations. You need to search the directory for configuration files with such content and configure as needed, following a similar pattern:

For HWM_PVV:
```
mm2: MemMap2 {
   map[00] = sdcard.data8
   map[01] = 40
}
```

For msx:
```
mm2: MemMap2 {
   map[00] = sdcard.ss
   map[01] = sdcard.data
}
```

For n8vem, on the VV55 ports:

```
miso: Register {
   read[0] = sdcard.miso
   read[1] = sdcard.miso
   read[2] = sdcard.miso
   read[3] = sdcard.miso
   read[4] = sdcard.miso
   read[5] = sdcard.miso
   read[6] = sdcard.miso
   read[7] = sdcard.miso
}

appmx: MemMap2 {
   map[0] = romdisk.data
   map[1] = miso
}

app: K580ww55 {
   portA = appmx
   portB = romdisk.lsb
   portC = romdisk.msb
   portC[0] = sdcard.mosi
   portC[5] = sdcard.sclk
   portC[E] = sdcard.ss
   portC[7] = appmx.offset
}
```

This configuration is used to emulate the SD interfaces (HWM_PVV, msx, and n8vem) in the "emu" emulator. You may need to adapt these configurations based on your specific needs or the specifics of the emulator you are using.

SD карта и SDOS для 8и битных ПК

У меня возникло желание упорядочить и собрать в одном месте всю информацию о системе SDOS. Честно говоря, я не предполагал, что SDOS разрастется до текущих маштабов, и сейчас сам иногда испытываю сложность в поиске информации о том или ином варианте сборки.

Что же такое SDOS?

SDOS это компактная консольная оболочка позволяющая читать, запускать на исполнение файлы с SD карты, отформатированной в FAT16, и записывать обратно на карту данные из ОЗУ компьютера. Это позволяет очень просто переносить файлы с 'больших' ПК в наши ретрокомпьютеры и обратно. В основе кода SDOS взят код уважаемого **b2m**, который я обрамил дополнительным функционалом и возможностями.

Код системы построен таким образом, что его очень просто можно адаптировать к любому ПК и к любому варианту аппаратного интерфейса и носителя (те, вообще говоря, это не обязательно должна быть SD карта, а может быть и HDD).

На данный момент SDOS:

- код самой SDOS занимает от 2 до 2.5КБ (может размещаться как в ПЗУ так и в ОЗУ), плюс для работы с FAT16 нужно еще 2КБ ОЗУ.

- собиралась и проверялась как минимум в эмуляторе emu, а в некоторых случаях и в реале на следующих ПК: Специалист std(реал) и MX2(реал), RK-86(реал), Апогей, Партнер-01, Galaksija(реал), Orion(реал), TRS-80(реал), ЮТ-88.

- поддерживает три базовых аппаратных интерфейса SD: HWM_PVV, msx, n8vem, и их производные варианты.

- поддерживает SD и SDHC карты, те карты с размером больше 4ГБ.

- поддерживает следующие базовые директивы:

- **CD** ИМЯкаталога - перейти в каталог с указанным именем;

- **DIR** - вывести список файлов и каталогов;

- **ИМЯфайла**.RKX(RKS для std, RKR для RK-86, GTP для Галаксии, RKO - Орион, CAS - TRS-80) запустить файл, при этом расширение можно не набирать, будет произведена автоподстановка;

- **R** ИМЯфайла.РАСШИРЕНИЕфайла,А ДРЕСкуда,СКОЛЬКОбайт - прочитать не запуская файл, начиная с указанного адреса в памяти и сколько байт пример: R TEST.BIN,0ACD,5FE0 - читает файл TEST.BIN в память начиная с адреса 0x0ACD и до адреса 0x0ACD+0x5FE0=0x6AAD. Ограничение - нет проверки на фактическую длину файла и запрошенную на чтение, те можно запросить прочитать больше чем размер файла, поведение не определено

- **W** ИМЯфайла.РАСШИРЕНИЕфайла,А ДРЕСоткуда,СКОЛЬКОбайт - записать в файл данные из памяти, начиная с указанного адреса в памяти и сколько байт пример: W TEST.BIN,0ACD,5FE0 - пишет в файл TEST.BIN из памяти начиная с адреса 0x0ACD и до адреса 0x0ACD+0x5FE0=0x6AAD. Ограничение - нет проверки на фактическую длину файла и запрошенную на запись, те можно запросить записать больше чем размер файла, поведение не определено. Записать больше чем существующий размер файла нельзя, если записать данных меньше чем размер файла, то размер файла не меняется и остается прежний

- **X** - перейти в монитор, из которого был запущен SDOS;

- **I** - запуск повторной инициализации карты;

только Galaksija:

- **WB** ИМЯфайла.РАСШИРЕНИЕ - запись BASIC программы в файл на SD в формате GTP;

только СпецециалистMX:

- **L** ИМЯфайла.РАСШИРЕНИЕфайла - прочитать данные файла формата RKX с SD в RAM диск MXa; пример: L TEST.BIN - читает файл TEST.BIN в память начиная с адреса 0x0000 и до адреса его длины, создает в RAM диске файл TEST.BIN с стартовым адресом и размером взятыми из 4х первых байт файла на карте, при том, что имя файла в хеадере отсутствовало (в 5м байте 0xE6).
Или: L TEST.BIN - читает файл TEST.BIN в память начиная с адреса 0x0000 и до адреса его длины, создает в RAM диске файл TEST_MX.HEX, с стартовым адресом и размером считанные из 4х первых байт файла на карте, и именем файла в хеадере TEST_MX.HEX.

- **S** ИМЯфайлаНаRAMдиске.РАСШИРЕН ИЕ,ИМЯфайлаНаSD.РАСШИРЕНИЕ - записать данные файла из RAM диска MXa на SD в формате RKX; пример: S TEST.BIN - читает файл TEST.BIN из RAM диска в память начиная с адреса 0x0000 и до адреса его длины, после записывает его на SD карту в существующий файл с таким же именем TEST.BIN, размер не изменяется, формат RKX!
Или: S TEST.BIN,TEST_MX.HEX - читает файл TEST_MX.HEX в память начиная с адреса 0x0000 и до адреса его длины, после записывает его на SD карту в существующий файл с именем TEST.BIN.

только Orion:

- **L** ИМЯфайла.РАСШИРЕНИЕфайла - прочитать данные файла формата RKO с SD в RAM диск; пример: L TEST$.RKO - читает файл TEST$.RKO в память начиная с адреса 0x0000 и до адреса его длины, создает в RAM диске файл TEST$ с именем, стартовым адресом и размером взятыми из хеадера RKO файла на карте.

- **S** ИМЯфайлаНаRAMдиске,ИМЯфайла НаSD.РАСШИРЕНИЕ - записать данные файла из RAM диска на SD в формате RKO. пример: S TEST$,TEST1$.BIN -читает файл TEST$ из RAM диска в память начиная с адреса 0x0000 и до адреса его длины, после записывает его на SD карту в существующий файл с таким же именем TEST1$.BIN, размер не изменяется, формат RKO!.

При выводе каталога по DIR печатается имя, расширение файла и его размер(что бы можно было использовать директивы R и W), а на директории пишется DIR в поле размера файла.

При запуске файла пишется стартовый и конечный адреса куда будет считан файл с карты.

Нажатие любой клавиши (пробела) при выводе по DIR приостанавливает вывода списка файлов.

Для СпециалистMX2 сделана возможность запускать не только файлы с расширением RKX, но и RKS, для чего в том же каталоге, где находится RKS файл нужно поместить файл монитора M2_C000.MON (взят из проекта Vinxru). Кроме того, есть возможность загружать произвольные мониторы.

И теперь масса ссылок, с упоминанием SDOS и SD интерфейса.

Началось все с этой темы по Специалисту с интерфейсом HWM, и тогда я думал, что на этом все закончится, да не тут то было.
Первая попытка упорядочить информацию по SDOS в рамках Специалиста.

Версия для РК-86 с интерфейсом HWM_PVV, и там же, дале с интерфейсом RK86_WW55_SD_HWM_PVV,
далее Апогей с оригинальным вариантом подключения карты, аналог интерфейса SD_n8vem. В этих

вариантах SD карта подключена совместно с ROM диском, не исключая его!
Партнер-01 .

Версия для Галаксии с интерфейсом HWM_PVV и с интерфейсом SD_n8vem .
Практическая реализация в этой теме.

Ют-88 с интерфейсом SD_n8vem.
Орион .
TRS-80 .

Схемы интерфейсов:

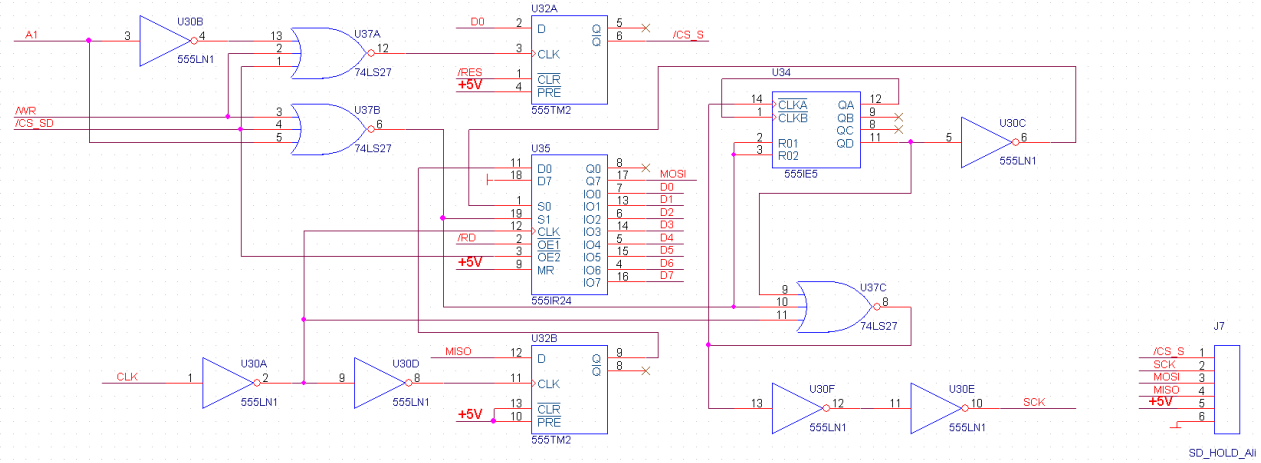
HWM_PVV на ИР8 и ИР9 или на портах ВВ55 .
HWM_PVV на ИР24 или здесь или здесь
HWM_PVV на ИР13

SD_n8vem или здесь или здесь или здесь

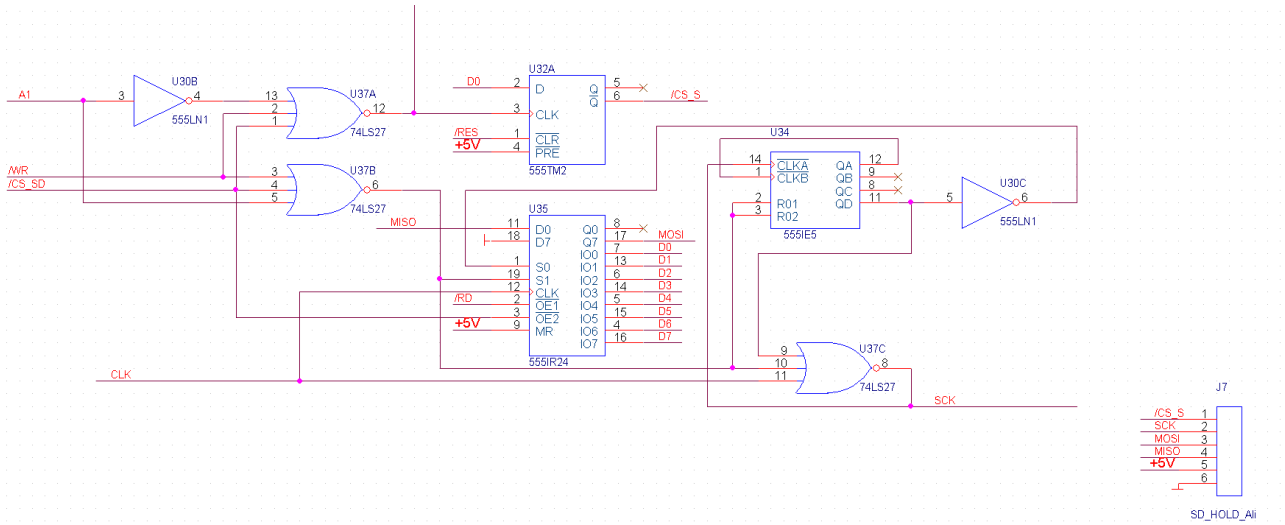
SD_msx я на реале не проверял, только в emu, но изначально код b2m был реализован для работы именно с этим интерфейсом и он использовался на реале в плис.

Хочу дополнить еще один момент. Согласно спецификации, при инициализации SD карты нужно использовать частоту тактирования SD_CLK не выше 400кГц, а по завершению инициализации переключаться на более высокую. Но как показала практика, современные карты спокойно инициализируются и работают на частоте 2-4МГц, что позволяет отказаться от переключателя скорости тактирования. Переключение скорости поддерживается только в интерфейсе на ПЛИС HWM и в первом варианте интерфейса HWM_PVV на дискретах.
Последний раз редактировалось PVV; 11.03.2020 в 23:17.

In addition to the SD interface circuits listed above, I would like to add one more circuit option, seen here. The entire SD interface is obtained using 5 ICs. Software fully compatible with the SD_HWM_PVV option.



As it turned out, this scheme can be further simplified without compromising functionality:



Инструкция и памятка для себя по SDOS:
Текущая версия SDOS_v8C

поддерживаются ПК: Специалист std и MX, RK-86, Galaksija, Orion, TRS-80, JupiterACE
поддерживает аппаратные интерфейсы SD: HWM_PVV, msx, n8vem

поддерживает следующие базовые директивы:
- CD ИМЯкаталога - перейти в каталог с указанным именем;
- DIR - вывести список файлов и каталогов;
- ИМЯфайла.RKX(RKS для std, RKR для RK-86 и GTP для Галаксии, RKO - Орион, CAS - TRS-80
)
  запустить файл, при этом расширение можно не набирать, будет произведена
  автоподстановка;
- R ИМЯфайла.РАСШИРЕНИЕфайла,АДРЕСкуда,СКОЛЬКОбайт - прочитать не
  запуская файл, начиная с указанного адреса в памяти и сколько
  байт (пример: R TEST.BIN,0ACD,5FE0 - читает файл TEST.BIN в память
  начиная с адреса 0x0ACD и до адреса 0x0ACD+0x5FE0=0x6AAD). Ограничение -
  нет проверки на фактическую длину файла и запрошенную на чтение, те
  можно запросить прочитать больше чем размер файла, поведение не
  определено;
- W ИМЯфайла.РАСШИРЕНИЕфайла,АДРЕСоткуда,СКОЛЬКОбайт - записать в файл
  данные из памяти, начиная с указанного адреса в памяти и сколько байт
  (пример: W TEST.BIN,0ACD,5FE0 - пишет в файл TEST.BIN из памяти начиная
  с адреса 0x0ACD и до адреса 0x0ACD+0x5FE0=0x6AAD). Ограничение - нет
  проверки на фактическую длину файла и запрошенную на запись, те можно
  запросить записать больше чем размер файла, поведение не определено.
  Записать больше чем существующий размер файла нельзя, если записать
  данных меньше чем размер файла, то размер файла не меняется и остается
  прежний;
- X - перейти в монитор, из которого был запущен SDOS;
- I - запуск повторной инициализации карты;

только Galaksija:
- WB ИМЯфайла.РАСШИРЕНИЕ - запись BASIC программы в файл на SD в формате GTP;

только СпецециалистMX:
- L ИМЯфайла.РАСШИРЕНИЕфайла - прочитать данные файла формата RKX с SD в RAM диск MXa;
- S ИМЯфайлаНаRAMдиске.РАСШИРЕНИЕ,ИМЯфайлаНаSD.РАСШИРЕНИЕ - записать данные файла из
  RAM диска MXa на SD в формате RKX;

только Orion:
- L ИМЯфайла.РАСШИРЕНИЕфайла - прочитать данные файла формата RKO с SD в RAM диск;
- S ИМЯфайлаНаRAMдиске,ИМЯфайлаНаSD.РАСШИРЕНИЕ, - записать данные файла из RAM диска на
  SD в формате RKO.

  При выводе каталога по DIR печатается имя, расширение файла и его
  размер(что бы можно было использовать директивы R и W), а на директории
  пишется DIR в поле размера файла.

При запуске файла пишется стартовый и конечный адреса куда будет считан
файл с карты.

добавлено 24.05.2017:   - для СпециалистMX(2)
- L ИМЯфайла.РАСШИРЕНИЕфайла - прочитать данные файла формата RKX с SD в RAM диск MXa
(пример: L TEST.BIN - читает файл TEST.BIN в память начиная
с адреса 0x0000 и до адреса его длины, создает в RAM диске файл TEST.BIN
с стартовым адресом и размером взятыми из 4х первых байт файла на карте,
при том, что имя файла в хеадере отсутствовало (в 5м байте 0xE6).
Или: L TEST.BIN - читает файл TEST.BIN в память начиная с адреса
0x0000 и до адреса его длины, создает в RAM диске файл TEST_MX.HEX,
с стартовым адресом и размером считанные из 4х первых байт файла на карте,

и именем файла в хеадере TEST_MX.HEX).

- S ИМЯфайлаНаRAMдиске.РАСШИРЕНИЕ,ИМЯфайлаНаSD.РАСШИРЕНИЕ, ИМЯфайлаНаSD опциональный параметр,
указывается только для варианта файлов с одинаковыми именами на RAM диске но разными расширениями - записать данные файла из RAM диска MXa на SD в формате RKX (пример: S TEST.BIN -
читает файл TEST.BIN из RAM диска в память начиная с адреса 0x0000 и до адреса его длины, после записывает его на SD карту в существующий файл с таким же именем TEST.BIN, размер не изменяется, формат RKX!
Или: S TEST.BIN,TEST_MX.HEX - читает файл TEST_MX.HEX в память начиная с адреса 0x0000 и до
адреса его длины, после записывает его на SD карту в существующий файл с именем TEST.BIN).

14.02.2018 SDOS_v8
сделал поддержку SDHC карт, есть отладочный вывод с типом обнаруженной карты и информации
 о наличии FAT16 на карте.
Добавил директиву I - запуск повторной инициализации карты.

21.02.2018 SDOS_V8.1
для Галаксии добавил директиву:
WB ИМЯфайла.РАСШИРЕНИЕ - запись BASIC программы в файл на SD в формате GTP.

01.03.2018 SDOS_V8.3
- L ИМЯфайла.РАСШИРЕНИЕфайла - прочитать данные файла формата RKO с SD в RAM диск Orion (пример: L TEST$.RKO - читает файл TEST$.RKO в память начиная
с адреса 0x0000 и до адреса его длины, создает в RAM диске файл TEST$
с именем, стартовым адресом и размером взятыми из хеадера RKO файла на карте.

- S ИМЯфайлаНаRAMдиске,ИМЯфайлаНаSD.РАСШИРЕНИЕ, - записать данные файла из RAM диска на SD в формате RKO (пример: S TEST$,TEST1$.BIN -читает файл TEST$ из RAM диска в память
начиная с адреса 0x0000 и до адреса его длины, после записывает его на SD карту в существующий файл с таким же именем TEST1$.BIN, размер не изменяется, формат RKO!).

05.03.2018 SDOS_V8.4
добавил поддержку TRS-80

01.06.2018 SDOS_V8.6
добавил чтение двух байт CRC после вычитки сектора, пакета данных в 512 байт

10.07.2018 SDOS_V8.7
добавил поддержку SD_WW55_n8vem

11.10.2019 SDOS_V8A
изменил размещение кода внутри самой SDOS и оптимизировал по размеру

04.03.2020 SDOS_V8C
первая попытка добавления IDE вместо SD на примере Ориона

{
формат заголовка файла RKX:
1) 2 байта -стартовый адрес;
2) 2 байта -конечный адрес (длина данных=конечный адрес-стартовый адрес);
3) опционально, 1+8+1+3=13 байт - признак автозапуска, имя и расширение файла в RAMFOS, между именем
и расширением ' ', автозапуск этого файла с его адреса загрузки 0 - нет, FF - да;
4) опционально, 1+8+1+3=13 байт -признак автозапуска, имя монитора с расширением,

между именем и расширением '.', автозапуск имеет приоритет над 3), без 3) не имеет смысла;

5) опционально, 1+8+1+3=13 байт еще один монитор (как пункт 4)), автозапуск имеет приоритет над 4);
6) 0xE6 - это 'маркер завершения' дескриптора RKX;
7) данные файла;
}

{
формат заголовка RKO:
8 байт имя
64 нулевых байт, потом 0E6h (синхробайты)
2 байта начало (обычно 0000)
2 байта конец (старший байт первый)
16 байт ORDOS-заголовок
(конец-начало-10h) данные
3 нулевых байта, потом 0E6h (синхробайты)
2 байта контрольная сумма
}

{
формат заголовка ORD:
0-7 - ИМЯ ФАЙЛА. МОЖЕТ СОДЕРЖАТЬ НЕ БОЛЕЕ 8 СИМВОЛОВ. ЕСЛИ ИМЯ СОДЕРЖИТ МЕНЬШЕ СИМВОЛОВ,
  СВОБОДНЫЕ ЯЧЕЙКИ ЗАПОЛНЯЮТСЯ ПРОБЕЛАМИ.
8-9 - НАЧАЛЬНЫЙ АДРЕС РАЗМЕЩЕНИЯ ПРОГРАММЫ ПРИ СЧИТЫВАНИИ ЕЕ ИЗ ДИСКА В ОЗУ - АДРЕС "ПОСАДКИ".
A-B - РАЗМЕР ФАЙЛА. В ЭТОТ ПАРАМЕТР ОГЛАВЛЕНИЕ ФАЙЛА(16 БАЙТ) НЕ ВХОДИТ.
C - БАЙТ ФЛАГОВ. В "ORDOS" V2.X ИСПОЛЬЗУЕТСЯ ТОЛЬКО БИТ D7. СОСТОЯНИЕ "1" УКАЗЫВАЕТ НА ТО,
  ЧТО ФАЙЛ ЗАЩИЩЕН ОТ УНИЧТОЖЕНИЯ. ОСТАЛЬНЫЕ БИТЫ ЗАРЕЗЕРВИРОВАНЫ ДЛЯ РАСШИРЕНИЯ. ИЗМЕНЕНИЕ
  СОСТОЯНИЯ БИТА D7 ПРОИЗВОДЯТ ВНЕШНИЕ ЗАГРУЖАЕМЫЕ ДИРЕКТИВЫ ОПЕРАЦИОННОЙ СИСТЕМЫ.
D-F - СЛУЖЕБНЫЕ ЯЧЕЙКИ СИСТЕМЫ.
}

{
CAS File Format for SYSTEM programs
----
binary file contains blocks of data

1st block: header: 256 times '00' and 'A5' synchron byte
2nd block: name: 55, and 6 character name from ASC 'A'-'Z'
data block: 3C NN LL HH DD DD DD ... DD CC
      where:
        3C: signature byte
        NN: length of the data block (0 means 256 bytes)
        HHLL: address of memory
        DD DD .. DD: data bytes
        CC: checksum
last block: 78 LL HH
      where HHLL is the start address of the program
}