VCPU R1/R2

VCPU programmer's documentation

V1.3.0

English translation: Siz

25/09/2024

1 Features

- Programmable high-capacity storage for CBM computers
- Partial 6502 CPU emulation
- Extended CPU functions
- Dedicated CPU instructions to implement communication with the computer
- Program memory can be used up to the size of file buffers (max. 1536 or 3840 BYTEs)
- Unified virtual hardware for different drive implementations
- Ability to handle large files
- System calls to access different functions of the drive
- All functions of the drive available to the CPU emulation
- Possibility of software implementation of existing fastloaders
- Etc.

1.1 Goals

The goal of the Flexible SD drive *firmware*[1] (FlexSD fw. in short) VCPU[2] extension is to make the SD2IEC[3] drive capable of executing downloaded programs like the classic Commodore disk drives. SD2IEC is a drive family with different hardware. With VCPU the executed program uses a unified (virtual) hardware so separate support is no longer needed. **Compatibility with existing older CBM drives is not a goal**! VCPU is not limited to regular disk images, it can use all of the functions of the drive so a much larger storage becomes available to the computer than the old disk drives. In addition any data transfer protocol can be implemented to fulfill the requirements of the programmer.

1 **Flexible SD drive firmware (FlexSD fw)** is an **sd2iec firmware** fork. (The original can be found at https://sd2iec.de) In this documentation the term firmware means **FlexSD firmware**.

2 VCPU: “Virtual CPU”. Most of these drives are built on a microcontroller (an integrated circuit containing a CPU, data- and program memory and peripherals – so in fact a complete computer) which can execute command only from ROM. It’s RAM can contain data only! The VCPU is a software CPU implementation running in the microcontroller ROM which executes the program copied to RAM.

3 In this documentation the name **SD2IEC** refers to the hardware. These are the same as the supported hardware by the original *sd2iec firmware*.

1.2 Differences compared to programming the original Commodore drives

Programming the drive's VCPU requires a bit different logic than the direct hardware programming of the Commodore drives. Accessing the direct storage device is limited. The usable commands the computer can send over the serial bus are the same but during VCPU execution the program "moves" to the drive memory where the results of the executed tasks are stored too. The only "near-hardware" task is communication with the computer; VCPU provides special instructions for this.

1.3 License

Both *FlexSD firmware* and it's VCPU extension are licensed under GPLv2. So using parts or whole of the firmware must be in accordance with this license. But this license does not apply to programs executed by the VCPU! The programmer can choose whatever license he/she wants for the executed programs[4].

4 The programmer must be aware that the program is inspected and processed by a software and there is no protection mechanism against deciphering them.

2 Commands available for the computer

To program the SD2IEC drive several new commands are required which makes the VCPU functions available to the computer. These commands (with the help of the KERNAL) can be executed in the usual way by sending them over the “command channel” and the responses can be read from the “error channel”. (The programming is the same as for the traditional CBM drives, details can be found in the well known documentations.)

The VCPU extension commands follows **“Zx”** form. To an unknown VCPU command the drive will respond with a “`35,SYNTAX ERROR,00,00`” error message.

2.1 “ZI”: Query VCPU information

The result is 5 BYTEs long:

1. VCPU version which is made of two parts: B5..0[5]: the version itself (currently `$1`[6] (**R1**) / `$2` (**R2**)), B7..5: device bus type. (Currently `$2`: CBM SERIAL, `$3`: CBM FAST SERIAL, `$4`: CBM SERIAL + PARALLEL or `$5`: CBM FAST SERIAL + PARALLEL. Reserved type codes: `$1`: IEEE-488, `$6`: TCBM.)
2. Size of “command channel” in BYTEs. This is usually 120, there is no lesser value present. The computer can write this many BYTEs to the drive in the “command channel”!
3. Size of “error channel” in BYTEs. This is usually 100, there is no lesser value present. The drive can send this many BYTEs to the computer in the “error channel”!
4. Number of data buffers. This is 6 or more (maximum 15). The drive will reserve this many buffers of 256 BYTEs. This is actually the memory size of the VCPU, the usable program memory. (6×256 = 1.5 KBYTEs, 15×256 = 3.75 KBYTEs.)
5. Last BYTE of VCPU I/O area, maximum number of I/O registers -1.

Returned VCPU version is currently `$41`, `$42`, `$62`, `$82` or `$A2`. (`%010` / `%011` / `%100` / `%101` and `%00001` / `%00010`.) If the drive does not have a VCPU extension it will return “`30,SYNTAX ERROR,00,00`” error message. In this response the first BYTE is always `$33` (ASCII code of “3”). VCPU version BYTE will never have a value of `$33` so this can be used easily to detect the presence of the expansion in the drive *firmware*. (Checking for the presence of VCPU this command is the recommended. The drive – normal / short – version string characters does not necessarily refer to VCPU support, they may change in the future!)

5 Designation of bits in a BYTE is “**B**” + number of bit. In case of multiple bits the range is specified! (**B5..0**) Numbering of the bits is the usual: the least significant bit is 0 (**B0**), the most significant is 7 (**B7**).

6 In this documentation the number systems follow the convention of 6502 *assemblers*: hexadecimal numbers are marked with prefix “**$**”, binaries are marked with prefix “**%**”. Numbers without prefix are interpreted in decimal number system.

2.2 “ZB”: Query data buffer state

The response is 2 BYTEs for all usable data buffers in the device:

1. Data buffer in use marker, bit 0 (B0) with value `%1` marks a used buffer, `%0` marks free. Other bits can be used in drive *firmware* but their meaning is irrelevant here. (Meaning of bits other than B0 can change in the future so their value must be ignored!)
2. The channel number belonging to the data buffer (Parameter 3 of OPEN statement, the “secondary address”) used by the computer. This is only valid when the buffer is in use. (B0 of the previous BYTE = `%1`)! Value is between 0..14 is the data buffer is used for normal file operations. Drive *firmware* can allocate buffers for special uses. In this case their channel number will be outside of this range.

The above two BYTEs are repeated in the response as many times as many data buffers the drive has.

2.3 “ZR”+ADDRLO+ADDRHI+LENGTH: Read data buffers as memory

This is the equivalent of the “`M-R`” command of the 15x1 drives but this one is limited to data buffers only so not all of the SD2IEC drive memory can be read. Size of memory read (with a single command) is limited by the size of the “error channel”. Attempting to read more will result in error! (“`35,SYNTAX ERROR,01,00`”) Omitting “`LENGTH`” parameter will result in reading 1 BYTE. If the read refers to an address outside the usable memory range it will result in an error too! (“`35,SYNTAX ERROR,02,00`”)

2.4 “ZW”+ADDRLO+ADDRHI+BYTE1+…: Write data buffers as memory

Similar to the “`M-W`” command of 15x1 drives. Memory beginning from the specified address will be written with as many BYTEs as many are present in the package. (The command does not have a size parameter unlike the “`M-W`” command!). Number of writable BYTEs are limited by the size of the “command channel”, the whole memory write command must fit into it! If an address outside the usable memory range is specified an error will be returned (“`35,SYNTAX ERROR,03,00`”). After a successful write the response will be “`00, OK,00,00`”.

2.5 “ZE”+ADDRLO+ADDRHI+…: Execute program using VCPU

With parameters after the “ZE” command not only the start address but all of the VCPU registers can be set. When no parameters specified (not even a start address) the execution will resume with the actual state from the actual position. Parameters can be found at command “ZC”. The response (and the possibility of query) depends on the executed program, it can return anything.

WARNING: the time elapsed between issuing the command “ZE” and the start of the VCPU program execution is not guaranteed, it can depend on the type of the drive and other circumstances. Because of this it’s recommended to implement some kind of synchronization which “notifies” the computer that the drive side code has started. (For example it can generate an impulse on some of the lines of the serial bus and the computer waits for that.)

2.6 “ZC”: Query VCPU state

This command returns the (current) state of the VCPU:

- **PCL** + **PCH**: two BYTEs for current PC position
- **A**: Accumulator
- **X**: X index register
- **Y**: Y index register
- **SR**: status register
- **SP**: stack pointer
- **SPH**: stack pointer upper bits (B15..8)
- **ZPH**: zero page upper bits (B15..8)
- **INT**: interrupt code
- **FUNCT**: code of the called function
- **LASTOP**: code of last executed instruction
- **RRL** + **RRH**: R address register (R2)[7]
- **U1RL** + **U1RH**: U1 address register (R2)
- **U2RL** + **U2RH**: U2 address register (R2)

In command “ZE” the parameters are from **PCL** to **ZPH** are these values in this order. If no address specified the execution will start from the address that is resulted here. **INT** / **FUNCT** / **LASTOP** values can be used for troubleshooting.

7 The functions marked with (R2) are available in the **VCPU-R2** version!

3 The VCPU

3.1 Major differences

VCPU consists of a software implemented 6502 emulator and a connected (also emulated) hardware. 6502 emulation is not complete, there are some missing and some additional functions:

- Cycle exact execution was not a goal; current implementation executes programs at about the speed of a 1 MHz 6502. (Some instructions are faster, others are slower.)
- "Undocumented" instructions of the original MOS 6502 are not supported! When these opcodes are encountered the execution will be interrupted with an error code. (However some of these opcodes are assigned to new instructions those will execute their proper function.)
- BCD mode is not implemented! `CLD` instruction is equivalent to `NOP`, `SED` is "illegal".
- Traditional interrupt handling is missing. `SEI` instruction is `NOP`, `CLI` is "illegal". But `RTI` is implemented, can be used in the usual way.
- Program execution is possible only from memory assigned to RAM! This memory is the address range `$0000..$05FF` of the emulated 6502 for 6 data buffers and `$0000..$0EFF` for 15 buffers. Jumping outside this range will interrupt the execution with an error code!
- `JMP ($xxFF)` instruction works as expected, the buggy operation of the original 6502 is not implemented. Target address of the indirect jump must reside in the RAM area too!!
- `BRK` instruction is used for system calls but those are not executed by the CPU emulation. Because of this the stack operations of `BRK` are not implemented. (Therefore system calls can be used even if **SPH** / **SP** is not set!)

3.2 VCPU extended functions

This functions are not supported by the original 6502!

- **SPH** register: the stack pointer (**SP**) of the 6502 is 8 bits, upper 8 bits are always `$01`. In VCPU this is stored in register **SPH**, so the stack can be relocated to an other memory page. Over- or underflow of the 8 bits **SP** does not modify **SPH**!
- **ZPH** register: the zero page of the original 6502 is always located at `$0000..$00FF`. In VCPU the upper 8 bits of the address can be changed so the zero page can be relocated to an other memory page. This affect the zero page addressing modes only! In case of changed **ZPH** the direct addressed instructions (for example: `LDA $0080`) always refer to the real address[8]!
- **RR** / **U1R** / **U2R** address registers (**R2**): to be used for fast subroutine calls.
- Handling of the CBM SERIAL, the serial bus is done by special "micro" instructions. By using them any kind of data transfer protocol can be programmed "by basic steps". The execution time for most of these instructions are fixed so time critical transfers can be implemented too.

8 This is especially important for *Y-indexed* addressing modes because usually the instructions does not have a `$zp,Y` addressing mode! But compilers will generate `$qwer,Y` addressing mode which – in case of relocated zero page – will not use the expected address.

- There are also special instructions for handling the CBM FAST SERIAL and PARALLEL ports.

3.3 Memory map for the emulated 6502

- `$0000..$05FF` / `$0000..$0EFF`: the memory buffers for file handles and RAM area for programs
- `$FC00..$FCFF`: the “command channel” of the drive can be read/written in this area but only in the length of the channel
- `$FD00..$FDFF`: the “error channel” of the drive can be read/written in this area but only in the length of the channel
- `$FE00..$FEFF`: I/O area, current length is 16 or 18 BYTEs

3.4 I/O registers

Address	Mode	B7	B6	B5	B4	B3	B2	B1	B0
$FE00	RO	BUS2	BUS1	BUS0	VCPU4	VCPU3	VCPU2	VCPU1	VCPU0
$FE01	RO	IOSIZ2	IOSIZ1	IOSIZ0	BNO4	BNO3	BNO2	BNO1	BNO0
$FE02	R/W	NEXT	PREV	DISPLY	-	-	-	DIRTY	BUSY
$FE03	RO	USW1	USW0	-	UNIT4	UNIT3	UNIT2	UNIT1	UNIT0
$FE04	R/W	FW diagnostics, not usable							
$FE05	R/W	FW diagnostics, not usable							
$FE06	RO	FSRVRDY	FSRVEN						
$FE07	W	Binary → decimal converter peripheral – Write binary data to convert decimal digits							
	R	Decimal value – least significant digit							
$FE08	W	Binary → decimal converter peripheral – Write binary data to convert ASCII digits							
	R	Decimal value – middle significant digit							
$FE09	W	-							
	R	Decimal value – most significant digit							
$FE0A	R/W	ATN out	SRQ out	-	-	-	-	-	-
$FE0B	RO	ATN in	SRQ in	-	-	-	-	-	-
$FE0C	R/W	-	-	-	-	-	-	DAT out	CLK out
$FE0D	RO	DAT in	CLK in	-	-	-	-	-	-
$FE0E	R	CLK in	-	-	-	-	-	-	-
	W	-	-	-	-	-	-	-	CLK out
$FE0F	R	DAT in	-	-	-	-	-	-	-
	W	-	-	-	-	-	-	-	DAT out

If the drive has a parallel port, the following registers are also present:

Address	Mode	B7	B6	B5	B4	B3	B2	B1	B0
$FE10	R/W	Parallel port – data register							
$FE11	R	HRWP	HRWFLAG	0	0	0	DRWPO	1	DRWPRB
	W	-	-	-	HRWFCLR	-		-	-

- $FE00: VCPU version number, read only. ($41/$42/$62, the same value that is present in the response of the "ZI" command.)
- $FE01: I/O area size (B765), memory size (B43210), read only.
- $FE02: State of SD2IEC buttons and LEDs, read/write. B0 = "BUSY LED", B1 = "DIRTY LED". %1: LED is on. B6 = "PREV BUTTON", B7 = "NEXT BUTTON". If an external display can be connected to the drive, then B5 = "DISPLAY BUTTON". %1: button is depressed.
- $FE03: Current device number of SD2IEC device (B43210 = 8..11 (..15)), read only. If there is a switch on the drive to configure the device number, then B76 = device number switches status. (The switches status and the device number do not necessarily have to be the same value; the device number can be set to a value independent of the switches status with the appropriate command! Also, for some hardware, the power-on state is shown here, not the current state!)
- $FE04..$FE05: Diagnostics, the function may change in the future, not usable!!
- $FE06: Drive firmware "flags" BYTE, read only. B6 = %1: Fast Serial receiver enabled, B7 = %1: Fast Serial data received. The other bits is not documented.
- $FE07: Binary → decimal conversion peripheral. The BYTE written here is converted to decimal digits (000...255), when read, the least significant digit of the converted value (0...9, '0'..'9') is located here.
- $FE08: Binary → decimal conversion peripheral. The BYTE written here is converted to ASCII characters ("000".. "255"), when read, the middle significant digit of the converted value (0..9, '0'..'9') is located here.
- $FE09: Binary → decimal conversion peripheral. When read, the most significant digit of the converted value (0..2, '0'..'2') is located here.
- $FE0A: ATN / SRQ lines control, read/write. B7 = ATN, B6 = SRQ. For write the drive of the lines can be turned on/off for read the set state can be read back not the current state of the lines!
- $FE0B: Current state of ATN / SRQ lines, read only! B7 = ATN, B6 = SRQ.
- $FE0C: CLK / DAT lines control, read/write. B1 = DAT, B0 = CLK. For write the drive of the lines can be turned on/off for read the set state can be read back not the current state of the lines!
- $FE0D: Current state of CLK / DAT lines, read only! B7 = DAT, B6 = CLK.
- $FE0E: Current state of CLK line, read only! B7 = CLK.
- $FE0E: CLK line control, write only! B0 = CLK.
- $FE0F: Current state of DAT line, read only! B7 = DAT.

- `$FE0F`: DAT line control, write only! B0 = DAT.
- `$FE10`: Parallel port data register, read/write.
- `$FE11`: Additional bits for parallel port data transmission, read/write.

At address `$FE02` the LEDs on the device can be controlled. Not all devices contain both LEDs! There are SD2IEC hardware with only 1 LED, that can be controlled by the bit of “BUSY LED”. The LEDs can be programmed easily but the system calls (file open/close/etc.) can change them the usual way so their state is not guaranteed to be kept. (It’s useful to leave their control to the drive *firmware* “in normal use”). In case of error the usual “flashing LED” won’t work during the execution of a VCPU program!

The same `$FE02` address can be used to query the state of the drive’s two buttons, and the state of the external display button (one of them), if available. During a VCPU program execution the original functions of the buttons does not work! There is SD2IEC hardware without buttons! Therefore the use of these buttons is not recommended. The occasional disk image swap is possible in a programmed way but it’s useful to do it in an automated way without user interaction.

In the range of `$FE0A`..`$FE0F` the control lines for the serial bus can be found. These registers won’t be expanded in the future, the unused bits are guaranteed to return the value `%0`, any value can be written to the unused bits! The handling of the serial bus is possible using these registers but not recommended. For this task the VCPU has dedicated instructions.

The `$FE10`/`$FE11` addresses exist only on devices with a parallel port for diagnostic purposes. Their use is not recommended, there are separate instructions for this task in the VCPU instruction set.

4 CBM serial bus

4.1 Hardware

The lines of the serial bus are at logical high level when not in use. This high level can be pulled low by any device (computer / peripheral) attached to the bus. High level is possible on any lines of the bus when all of the devices turn off the pull low! Because of this in this document the term “driving the line” always means that the device pulls the line low and “releasing the line” means that the pulling low is switched off. In the latter case the high level can be present only when all of the devices “releases” that. (None of the devices can “force” a line to high level.)

Parallel port operation is similar to serial bus lines: if either side (computer / peripheral) sets a bit low, the line will be low anyway, the other side cannot "force" a line high[9]. For this reason there is no separate data direction register on the drive side, if data reception is required, all parallel port bits must be set to `%1` (`$FF` must be written to the port).

4.2 Software

Software can handle the lines of the serial bus as a bit of a peripheral register by. The “level” of the given line is shown by the appropriate peripheral bit. “Driving” and “releasing” the line is controlled by toggling a bit of a different peripheral. On the SD2IEC side the peripheral bit always shows the actual logical level of the lines (for a low level line `%0`, `%1` for high level), controlling the output is done the same way (`%0` pull low, `%1` releases).

9 For VIC20, one of the VIA "B" port is connected to USER PORT, this port can drive high levels with higher power. For this reason, the data direction on VIC20 should only be switched to output when $FF is set as parallel port data on the drive side!

4.3 Lines

Serial bus has 4 lines relevant for programming. These (and their original functions) are:

- SRQ: Service ReQuest. Not used in its original function, where this line should be driven by peripherals, it's input on computer. In 1541 this line is not connected. In VIC20 one of the VIAs receive it. The CPU can't read it's real level but the change can generate an interrupt. In the C64 it work the same: one of the CIAs receive it and an interrupt can be generated. It's not connected in the C16 / C116 / plus/4 computers. On the C128 the "fast serial bus" mode uses it. Currently the drive *firmware* provides basic support for it via the VCPU[10]. With C128, in "fast serial" mode possible to send/receive data. Warning: for some SD2IEC drives controlling the line is not implemented (missing the hardware)!
- ATN: ATtentioN: call for device addressing. This line is controlled by the computer, it's input on peripherals. At low level the device addressing is done, this is the time when the computer selects the peripheral to communicate to. In 1541 this is input only it's change controls turning on driving DAT. SD2IEC – unlike CBM peripherals – can drive it and it can be controlled by VCPU. On VIC20, C64, C16 / C116 / plus/4 and C128 the line is output only, the state of the line cannot be read back! Driving by SD2IEC can be useful when the communication is directly between peripherals without including a computer.
- CLK: CLocK: the "clock" of the data transfer, this helps to synchronize the transferred bits. Can be driven by both computers and peripherals.
- DAT: DATa: the transferred data bit. Can be driven by both computers and peripherals.

If the drive has a parallel data port, the above signals are supplemented by the following:

- D0..D7: 8-bit data bus, can be driven/read by computer and peripherals.
- HRWP: Host R/W Pulse: output on the computer side, input only on the drive. The computer uses this to indicate when it has written new data to the 8-bit bus or read data sent to it from the 8-bit bus (VIA CB2 line for VIC20, CIA PC(2) for C64/C128, ACIA RTS for plus/4).
- DRWP: Device R/W pulse: output on the drive side, input on the computer. This is used by the drive to indicate when it has written new data to the 8-bit bus or read data sent to it (VIA CB1 line for VIC20, CIA FLAG(2) for C64/C128, ACIA DSR for plus/4).

10 For drives with larger program memory, KERNAL-level support is also possible!

5 VCPU instruction set

The VCPU instructions can be divided to 3 groups: the original 6502 instructions, the extended functions' instructions and the serial bus handler "micro" instructions. The documentation for the instructions in the first group can be found in the well known documentations about programming the 6502.

5.1 Instructions of the extended functions

Mnemonic	Op-code	Registers[11]	Description
TYSPH	$D4	SPH	Transfer Y to SPH register
TSPHY	$F4	Y,N,Z	Transfer SPH to Y register
LDSPH #$xx	$22 $xx	SPH	LoaD SPH register

- TYSPH: moves value of register **Y** to register **SPH**
- TSPHY: moves value of register **SPH** to register **Y**
- LDSPH #$xx: loads the value to register **SPH**

TYZPH	$44	ZPH	Transfer Y to ZPH register
TZPHY	$54	Y,N,Z	Transfer ZPH to Y register
LDZPH #$xx	$02 $xx	ZPH	LoaD ZPH register

- TYZPH: transfers value of register **Y** to register **ZPH**
- TZPHY: transfers value of register **ZPH** to register **Y**
- LDZPH #$xx: loads the value to register **ZPH**

Registers **SPH** / **ZPH** must point to RAM area! On an attempt to write a value that points outside of RAM the program execution is interrupted with an error code.

The following instructions are only available in **VCPU R2**:

TYXU1	$CB $02	U1R	Transfer Y:X registers to U1R
TU1YX	$CB $03	X,Y	Transfer U1R to Y:X registers
TYXU2	$CB $04	U2R	Transfer Y:X registers to U2R
TU2YX	$CB $05	X,Y	Transfer U2R to Y:X registers
TYXRR	$CB $06	RR	Transfer Y:X registers to RR
TRRYX	$CB $07	X,Y	Transfer RR to Y:X registers

11 **N**, **V**, **B**, **D**, **I**, **Z**, **C** characters denote the bits of the status register, **PC**, **A**, **X**, **Y**, **SP**, **SPH**, **ZPH**, **RR**, **U1R**, **U2R** refers to CPU registers.

PSHRR	$CB $01	SP	Push RR to Stack
PULRR	$CB $00	SP,RR	Pull RR from Stack

- TYXU1: Moves the **Y:X** register pair to the **U1R** address register
- TU1YX: Moves the **U1R** address register to the **Y:X** registers
- TYXU2: Moves the **Y:X** register pair to the **U2R** address register
- TU2YX: Moves the **U2R** address register to the **Y:X** registers
- TYXRR: Moves the **Y:X** register pair to the **RR** address register
- TRRYX: Moves the **RR** address register to the **Y:X** registers
- PSHRR: Saves the **RR** address register to the stack
- PULRR: Restores the **RR** address register from the stack

These registers (**U1R**, **U2R**, **RR**) can only contain a valid memory address! If the address is invalid, the program will abort with an error code! The **Y** register contains the upper part of the address (B15..8), the **X** register contains the lower part (B7..0). The instructions for using registers are listed in the following group.

5.2 Serial bus handling “micro” instructions

Most of these instructions do not change the status register’s bits! Execution times are fixed, the unit is 1 clock period of the 1MHz 6502 in the 1541. (1 6502 clock cycle is “1T”.)

Mnemonic	Op-code	Time	Registers	Description
UWATL	$43	1.5T+	–	Wait for serial ATN line Low
UWATH	$53	1.5T+	–	Wait for serial ATN line High
UWCKL	$23	1.5T+	–	Wait for serial CLK line Low
UWCKH	$33	1.5T+	–	Wait for serial CLK line High
UWDTL	$03	1.5T+	–	Wait for serial DAT line Low
UWDTH	$13	1.5T+	–	Wait for serial DAT line High

- UWATL: Waits for serial bus line ATN low level
- UWATH: Waits for serial bus line ATN high level
- UWCKL: Waits for serial bus line CLK low level
- UWCKH: Waits for serial bus line CLK high level
- UWDTL: Waits for serial bus line DAT low level
- UWDTH: Waits for serial bus line DAT high level

These instructions wait for the desired level of the serial bus lines. If the line is already at the waited level before beginning of the instruction execution then execution time will be 1.5T. Waiting will be finished in less than 1T after the line reached the watched level and the execution continues.

USATL	$C3	1.5T	-	Set serial ATN line to Low
USATH	$D3	1.5T	-	Set serial ATN line to HighZ
USCKL	$A3	1.5T	-	Set serial CLK line to Low
USCKH	$B3	1.5T	-	Set serial CLK line to HighZ
USDTL	$83	1.5T	-	Set serial DAT line to Low
USDTH	$93	1.5T	-	Set serial DAT line to HighZ

- USATL: Sets serial bus line ATN to low
- USATH: Releases serial bus line ATN
- USCKL: Sets serial bus line CLK to low
- USCKH: Releases serial bus line CLK
- USDTL: Sets serial bus line DAT to low
- USDTH: Releases serial bus line DAT

These instructions control the serial bus lines. “Releasing the line” is done only on the SD2IEC side, other devices can still hold it at low level!

UCLDL	$0B	1.5T	-	Set serial CLK to Low / DAT to Low
UCLDH	$1B	1.5T	-	Set serial CLK to Low / DAT to HighZ
UCHDL	$2B	1.5T	-	Set serial CLK to HighZ / DAT to Low
UCHDH	$3B	1.5T	-	Set serial CLK to HighZ / DAT to HighZ

- UCLDL: Sets serial bus lines CLK and DAT to low
- UCLDH: Sets serial bus line CLK to low and releases DAT
- UCHDL: Releases serial bus line CLK and sets DAT to low
- UCHDH: Releases serial bus lines CLK and DAT

With these instructions the serial bus CLK and DAT lines can be controlled in a single step.

UATCK #$xx	$6B $xx	2T	-	Copy **A** register bit To CLK line
UCKTA #$xx	$7B $xx	2T	A	Copy CLK line to **A** register bit
UATDT #$xx	$4B $xx	2T	-	Copy **A** register bit To DAT line
UDTTA #$xx	$5B $xx	2T	A	Copy DAT line To **A** register bit

- UATCK #$xx: Drives/releases the CLK line based on the selected bit of register **A**

- `UCKTA #$xx`: Copies the state of CLK line to the selected bit of register **A**
- `UATDT #$xx`: Drives/releases the DAT line based on the selected bit of register **A**
- `UDTTA #$xx`: Copies the state of DAT line to the selected bit of register **A**

The parameter of the instructions is a bitmask where only a single bit can have the value `%1`! Reading CLK / DAT the value of the line is copied to that bit of accumulator where the %1 is present in the bitmask parameter. On write driving/releasing the state of the line will be set based on the bit of accumulator where the bitmask value is %1. Attention: VCPU **SR** bits does not change on accumulator value change!

`UATCD #$xx,#$yy`	`$8B $xx $yy`	`2.5T`	`-`	Copy **A** register bits To CLK / DAT
`UCDTA #$xx,#$yy`	`$9B $xx $yy`	`2.5T`	`A`	Copy CLK / DAT line To **A** register bits

- `UATCD #$xx,#$yy`: Controls the CLK/DAT lines based on bits of register **`A`**
- `UCDTA #$xx,#$yy`: Copies the state of CLK/DAT lines to the selected bits of register **`A`**

The parameters of the instructions are two bitmasks, both of them with a single %1 bit! The first bitmask belongs to the CLK line and the second to DAT. On read the state of CLK will be written to the bit of accumulator where the %1 value is present in the first bitmask and the state of DAT likewise with the second bitmask. On write the first bitmask controls driving/releasing CLK and the second control DAT. Attention: VCPU **`SR`** bits are not affected even when value of accumulator is changed!

These instructions are restricted to parameter values where only one bit is `%1`. (So `$01`, `$02`, `$04`, `$08`, `$10`, `$20`, `$40`, `$80`.) Result of other bit combinations is “undefined”. (On read when multiple bits are `%1`, the selected line will be copied to all bits with `%1` value in the mask. Probably this won’t change in the future. On write when multiple bits are `%1` then some logical combination of accumulator bits will determine the state of the line that can depend on the hardware version of SD2IEC so it’s not recommended to use “illegal” bitmasks.) On instructions that read/write two lines the two bitmasks cannot be the same! (On read it’s “undefined” which line’s state will be copied to accumulator. On write both lines’ state will be set based on the same bit of accumulator, most probably this won’t change in the future.)

ULBIT	$DB	2T	N,V,Z	Lines BIT test

- ULBIT: Copies the current state of the serial lines to the bits of the status register

This instruction copies the current state of the serial lines to the bits of the status register: ATN goes to V, CLK goes to N, and DAT goes to Z. The conditional jumps after the instruction will do the jump based on the state of the checked line. (BVS / BVC: ATN line high / low, BMI / BPL: CLK line high / low, BEQ / BNE: DAT line high / low.)

USND1	$63	?T		Send **A** register to DAT, 1 bit mode
URCV1	$73	?T	A	Receive from DAT to **A** register, 1 bit mode
USND2	$E3	?T		Send **A** register to CLK/DAT, 2 bit mode
URCV2	$F3	?T	A	Receive from CLK/DAT to **A** register, 2 bit mode

- USND1: 1 bit synchronized data send
- URCV1: 1 bit synchronized data receive
- USND2: 2 bits synchronized data send
- URCV2: 2 bits synchronized data receive

The “1 bit” instructions implement sending/receiving of 1 BYTE synchronized data using CLK / DAT lines. Synchronization is done by line CLK which is toggled by the computer and the drive responds to it’s changes. Because of this the execution time is undefined, depends on the speed of the computer:

- The USND1 instruction is for sending data. At first it switches off driving of line CLK then waits for it’s high level. When CLK is high it copies the value of bit 0 of register **A** to DAT. After that it wait for low level of line CLK. When the computer received B0 it switches CLK to low. In response the drive copies bit 1 of register **A** to DAT and waits for CLK to be high. When the computer received B1 it releases CLK. Then the drive puts the next bit to DAT and so on. At the end of the BYTE the computer sets CLK to low and the drive puts bit 7 of register **A** to DAT and the instruction terminates and the next one can begin.
- The URCV1 instruction is for receiving data. At first it switches off driving of lines CLK / DAT then waits for CLK low level. The computer copies B0 of the data to be sent to DAT and switches CLK to low. In response the drive reads DAT to bit 0 of register **A** then waits for CLK high. The computer sets B1 to DAT then toggles CLK to high. Then the drive reads DAT to bit 1 of register **A** and so on. At the end of the BYTE the computer copies bit 7 to DAT and releases CLK then the drive reads B7 to register **A** and the instruction terminates and the next one can begin.

The “2 bits” instructions use lines ATN / CLK / DAT to implement synchronized sending / receiving of a whole BYTE. The synchronization is done with line ATN which is toggled by the

computer and the drive reacts to it's changes. Because of this the execution time is undefined, depends on the speed of the computer:

- The USND2 instruction is for sending data. At first it switches off driving ATN then waits for it's high level. When ATN is high it copies bit 1 of register **A** to DAT and bit 0 to CLK. Then waits for ATN low level. When the computer received B1/B0 it toggles ATN to low. In response the drive copies bit 3 of register **A** to DAT and bit 2 to CLK then waits for ATN high level. When the computer received B3/B2 it releases ATN. Then the drive puts the next bit pair to DAT/CLK and so on. At the end of the whole BYTE the computer toggles ATN to low and in response the drive puts bits 6 / 7 of register **A** to CLK/DAT and the instruction terminates and the next one can begin.
- The URCV2 instruction is for receiving data. First it switches off driving ATN / CLK / DAT lines and waits for ATN low. Computer puts B0 of data to DAT and B1 to CLK then sets ATN to low. The drive reads DAT to bit 0 of register **A** and CLK to bit 1 and waits for ATN high. Computer sets B2 to DAT and B3 to CLK then sets ATN to high. The drive reads DAT to bit 2 of register **A** and CLK to bit 3 and so on. At the end of the BYTE the computer copies bits 6 / 7 to DAT / CLK lines and releases ATN. Then drive reads DAT to B6 of register **A** and CLK to B7 and the instruction terminates and the next one can begin.

The following FSxxx (Fast Serial handling) instructions are only available on drives supporting the CBM FAST SERIAL bus (see DOS command "ZI"):

FSTXB	$8F	40T		Send **A** register to DAT/SRQ, 1 bit fast serial mode
FSRXB	$9F	1.5T+	A	Receive from DAT/SRQ to **A** register, 1 bit fast serial mode
FSRXC	$AF	1.5T	V	Check Fast Serial Received flag
FSRDS	$CB $10			Disable Fast Serial receiver
FSREN	$CB $11			Enable Fast Serial receiver

- FSTXB: Sends the contents of register **A** via Fast Serial lines (DAT+SRQ)
- FSRXB: Loads data received on Fast Serial lines (DAT+SRQ) into register **A** (**SR** bits do not change). If no data is received, it waits until the next one arrives!
- FSRXC: Checks if there is new data received on Fast Serial lines (SRQ+DAT). The result is stored in bit **V** of the status register (%1: new data present, %0: none)
- FSRDS: Disable Fast Serial data receiver
- FSREN: Enable Fast Serial data receiver

These instructions can be used to send/receive data via the "Fast Serial" bus. The drive has the necessary peripherals emulated in software, so there are some restrictions on its use. Data reception can be enabled/disabled with the FSREN / FSRDS commands (disabled by default). If data reception is enabled, the speed of the VCPU program execution during bit reception is drastically reduced! If a full BYTE is received, the received data is transferred to a temporary 1 BYTE buffer, the FSRXB instruction copies the contents of this buffer to register **A**. This data can be read at any

time until the next full BYTE is received. Attention: VCPU **SR** bits are not modified! If no data is received, the FSRXB instruction waits until the next one is received. The FSRXC instruction is used to check whether there is received (but not yet read) data in the buffer. Warning: if data reception is not enabled, the FSRXB instruction will never ended! Data reception will be error-free only if the drive keeps the DAT line in the released state!

FSTXB instruction can be used to send data at any time (regardless of whether reception is enabled). When the instruction is executed, data reception is disabled, the contents of register **A** are transmitted using the DAT+SRQ lines, and at the end of the transmission, data reception is enabled again if it was enabled before the instruction was executed. Therefore, the direction of communication does not (cannot) need to be switched (on the drive side), the process is automatic. After execution of the instruction, the value of the last data bit sent (register **A** B0) remains on the DAT line! The DAT line can be switched at any time after the command has been executed (no waiting is required), the data sent is received by the other side without damage.

The following PPxxx (parallel port handling) instructions are only available on drives supporting the PARALLEL bus:

PPDRD	$17	2T	A	Read data from parallel bus to **A** register, clear HRWFLAG
PPDWR	$27	2T		Write data to parallel bus from **A** register, clear HRWFLAG
PPACK	$07	2.5T		Send ACK pulse on DRWP line
PPWAI	$37	1.5T+		Wait ACK pulse on HRWP line
PPWDW	$47	2.5T		Wait ACK pulse on HRWP line, then write data to parallel bus from **A** register, clear HRWFLAG

- PPDRD: Reads the current state of the parallel port into register **A** (**SR** bits do not change) and clears HRWFLAG
- PPDWR: Write the value of register **A** to the parallel port and clear HRWFLAG
- PPACK: Generates a pulse on the DRWP line to the computer
- PPWAI: Waiting for a pulse on the HRWP line from the computer. The drive hardware "remembers" any pulse during the program run and set to the HRWFLAG. For this reason, it is not necessary for the pulse to occur while this instruction is running! If a pulse has already been received, the run time is 1.5T.
- PPWDW: It waits for a pulse on the HRWP line, and if it has arrived (even earlier), it writes the value of register **A** to the parallel port and clears HRWFLAG

UINDB $offs	$AB $of	1.5T/2T	X,Y	INcrement **X**, Decrement **Y**, Branch if no cy.
UDEDB $offs	$BB $of	1.5T/2T	X,Y	DEcrement **X**, Decrement **Y**, Branch if no cy.

- UINDB $offs: increment **X**, decrement **Y** and jump when **Y** did not became $FF
- UDEDB $offs: decrement **X**, decrement **Y** and jump when **Y** did not became $FF

These instructions increase or decrease the value of register **X**. Then decrease the value of register **Y**. When that underflow from $00 to $FF the execution continues with the next instruction. In this case the execution time is 1.5T. When **Y** did not become $FF execution will jump to the parameter of the instruction and the whole execution time will be 2T. (The destination address is calculated in the same way as for all 6502 conditional jumps: the parameter is treated as a signed value in the range of -128..+127 and added to PC). Crossing a 256 BYTE page boundary does not affect the execution time. Attention: the VCPU **SR** bits are not affected by the execution of these instructions!

UDEL1	$EA	1T	-	1T DELay
UDELY #$xx	$FB $xx	1.5T+	-	DELaY

- UDEL1: 1T delay
- UDELY #$xx: 1.5T + $xx × 0.5T delay

UDEL1 instruction is 1T delay. (This is the opcode of NOP on the original 6502, execution time: 1T). UDELY is a delay instruction where the time is specified by the fixed value in the parameter. The shortest time is 1.5T. That is extended with the parameter value × 0.5T.

There are four additional data movement instructions:

LDA $zp,X	$B5 $zp	2T	A,N,Z	LoaD **A**ccumulator from zero page **X** indexed
STA $zp,X	$95 $zp	2T	-	STore **A**ccumulator to zero page **X** indexed
PLA	$68	1.5T	SP,A,N,Z	PulL **A**ccumulator from stack
PHA	$48	1.5T	SP	PusH **A**ccumulator to stack

The first two instructions are the original 6502 **X** indexed zero page read / write instructions but their execution time is 2T. As the VCPU zero page can be relocated anywhere (within the memory) they can read / write the whole memory. The other two instructions are the original 6502 stack read / write instructions but their execution time is 1.5T. As the VCPU stack can be relocated they can access the whole memory.

The following instructions are only available in **VCPU R2**:

USER1	$1F	1.5T	RR,PC	Call USER1 vector
USER2	$2F	1.5T	RR,PC	Call USER2 vector
USERR	$0F	1T	PC	Return from USERx call

- USER1: fast call to a subroutine at an address in the **U1R** address register
- USER2: fast call to a subroutine at an address in the **U2R** address register
- USERR: Return from USER1 / USER2 call

These instructions can be used to "fast" subroutine calls. When the USER1 / USER2 instruction is executed, the current value of the PC (the address after the USERx instruction) is saved in the **RR** address register, and the **U1R** / **U2R** address register is written to the **PC** and the program execution continues from there. The USERR instruction causes the **PC** saved in the **RR** to be restored and the program run to continue at the address after the call. Compared to a traditional subroutine call (JSR $qwer), it has the advantage of a fixed and short execution time and the length of the calling instructions, which is 1 BYTE. However, the return address is not (automatically) saved in the stack, so you cannot nest these calls! Instructions for setting registers are given at the previous group.

5.3 Call drive *firmware* functions

Mnemonic	Op-code	Registers	Description
`BREAK #$xx`	`$00 $xx`	`A,X,Y`	System Call

- `BREAK #$xx`: call system function number `$xx`

This is the `BRK` instruction of the original 6502. The instruction and the following byte is processed by directly drive *firmware* instead of the VCPU. The VCPU state is saved and the system call is processed.

6 System calls

With the `BREAK #$xx` instruction various drive *firmware* functions can be accessed. The parameter is the number of the system function. The program execution will be suspended when the function requires stopping the CPU emulation. For other tasks the emulated 6502 code execution continues from the address after the parameter of the `BRK` + parameter instruction. The system call will put its return values in VCPU **`A`** / **`X`** / **`Y`** registers so their contents are not preserved! The bits of the status register are not set by the value of any of the registers, the returned values must be checked!

6.1 Codes of system calls

Code	Referenced name	Description
$00	SYSCALL_EXIT_OK	Exit with messages “00, OK,00,00”
$01	SYSCALL_EXIT_SETERROR	Exit with selected message
$02	SYSCALL_EXIT_FILLEDERROR	Exit with “error channel” filled
$03	SYSCALL_EXIT_REMAIN	Exit without modifying the “error channel”
$11	SYSCALL_DISABLEATNIRQ	Disables the ATN active VCPU exit
$12	SYSCALL_ENABLEATNIRQ	Enables the ATN active VCPU exit
$13	SYSCALL_SETFATPARAMS	Set FAT file handling parameters
$21	SYSCALL_DIRECTCOMMAND	Execute command from “command channel”
$22	SYSCALL_DIRECTCOMMAND_MEM	Copy command from memory to “command channel” and execute it
$23	SYSCALL_OPEN	Open file (name in “command channel”)
$24	SYSCALL_OPEN_MEM	Open file (name in memory)
$25	SYSCALL_CLOSE	Close file
$26	SYSCALL_CLOSEALL	Close all open files/channels
$27	SYSCALL_REFILLBUFFER	Refill buffer for reading / buffer flush for write
$28	SYSCALL_GETCHANNELPARAMS	Query channel parameters
$31	SYSCALL_CHANGEDISK	Request change disk (-image)

6.2 Description of system calls

- $00: SYSCALL_EXIT_OK: Terminate VCPU program execution. The contents of the “error channel” will be “00, OK,00,00” and the computer will be able to handle the peripheral in the usual way.
- $01: SYSCALL_EXIT_SETERROR: Terminate VCPU program execution. The contents of the “error channel” will be set to the required message! Before the system call register **A** must contain the error code and registers **X** / **Y** must contain the track / sector numbers. Only existing error codes are allowed!
- $02: SYSCALL_EXIT_FILLEDERROR: Terminate VCPU program execution. The “error channel” must be filled and **X** must contain the number of bytes put in error channel. After that the system function can be called.
- $03: SYSCALL_EXIT_REMAIN: Terminate VCPU program execution. Without changing the contents of the “error channel”. (When a previous function returned with error using this function to terminate VCPU code execution the computer will be able to query the error in the usual way)
- $11: SYSCALL_DISABLEATNIRQ: The default behavior is to interrupt VCPU code execution when ATN goes active. This can be switched off with this system call when the ATN is required for other purposes.

- $12: SYSCALL_ENABLEATNIRQ: Turns back on the interrupt of VCPU code execution for ATN active.
- $13: SYSCALL_SETFATPARAMS: drive *firmware* reads / writes files in 254 byte block in case of the FAT filesystem, just like as in a disk image. This system call can change this behavior. Before the call register **X** must contain the starting position in within the data buffer (default is 2, this is the start for loaded data), and register **Y** contains the number of bytes to be read in one step (default value is 254). 0 means 256 BYTEs! After the system call registers **X** / **Y** will contain the values set. If the starting position plus BYTE count is higher than the size of the data buffer (256), the setting will not be applied! After return the registers will contain the actual values so with “bad” parameters the current settings can be queried.
- $21: SYSCALL_DIRECTCOMMAND: The computer can communicate the tasks to the drive via the “command channel”. (Delete file, change directory, etc.) VCPU can execute commands in the same way: the “command channel” is mapped in the memory of the 6502. The commands bytes must be written here directly. The number of bytes in the “command channel” must be set in register **X** and the system function can be called. Drive *firmware* will try to execute the command then returns to the VCPU program execution. After the system call register **A** will contain the error code from the “error channel” and register **X** will contain the number of bytes in the “error channel”.
- $22: SYSCALL_DIRECTCOMMAND_MEM: This command is the same as the previous one (number $21) but before execution the command will be copied from the VCPU memory to the “command channel”. Before call register pair **ZPH**:**Y** must be set to the starting address of the command and register **X** must be set to the number of bytes to be copied (the length of the command). Return values are the same as for command $21.
- $23: SYSCALL_OPEN: Open file. The “command channel” must contain the name and open mode of the file, register **X** must be set to the length of the filename. Register **A** must be set to the number of the requested channel which must be between 0..14. After return register **A** will contain the error code from the “error channel”, register **X** will contain the number of bytes written to the “error channel”. Important: new error code / error message will only be set in case of an actual error! In the case of error-free execution, the contents of the “error channel” will not be changed, and invalid error code ($FF) will be set in register **A**. Register **Y** will contain the *number* of the data buffer associated with the file. This value is the upper 8 bits of the VCPU memory. For example number $02 means that the buffer is at $0200..$02FF in the memory! If the buffer allocation fails or some error occurs during execution, $FF is written to the register. (If the **Y** register is $FF, it is worth checking the **A** register / "error channel".) Additionally in the “error channel” memory ($FDxx) three additional data can be found: starting position[12] of data in the buffer ($FD60), position of the last used BYTE ($FD61), and a flag signaling if there is data left in the file ($FD62), where if $00 is found, there is still data to read from the file.
- $24: SYSCALL_OPEN_MEM: This command is the same as the previous one (number $23) but before execution copies the file name from the VCPU memory to the “command channel”. Before call the address of the memory must be set in register pair **ZPH**:**Y**, in register **X** the number of bytes to be copied (length of the filename). Register **A** must contain

12 This value is the position of the "first" data not yet used from the buffer. If the VCPU program retrieves the data of a previously opened file (see $28: SYSCALL_GETCHANNELPARAMS), and this file has been read from KERNAL, this parameter returned is the position of the first unused data, not the first BYTE of the data in the buffer!)

the number of the requested channel just like with command $23. Returned values are the same as there.

- $25: SYSCALL_CLOSE: Close File / channel. Register **A** must contain the channel number specified for OPEN then call the system function. In case of successful execution register **A** will contain $00.
- $26: SYSCALL_CLOSEALL: Close all open files / channels. There are no parameters and return values.
- $27: SYSCALL_REFILLBUFFER: For file read this system call can be used to request loading of the next set of bytes. Register **A** must contain the number of the channel specified for OPEN the call the function. The returned parameters are the same as for commands $23 / $24: SYSCALL_OPEN (_MEM). For file write when the data buffer is full or no more bytes need to be written register **X** must contain the starting position in the buffer and register **Y** must contain the position of the last used byte. Register **A** is the number of channel used for OPEN. The returned parameters are the same as for command $23 / $24: SYSCALL_OPEN (_MEM).
- $28: SYSCALL_GETCHANNELPARAMS: Query channel parameters. Register **A** must be set to the number of the channel to be queried. Returned parameters are the same as for commands $23 / $24: SYSCALL_OPEN (_MEM).
- $31: SYSCALL_CHANGEDISK: “Disk change” function of drive *firmware*. Using this function the disk images can be swapped based on the contents of an external swap list file. Register **X** must be filled with the number of the disk image. Register **A** will be set to $00 on successful disk change. To use this function the swap file must be set for drive *firmware* (AUTOSWAP.LST). When the user chooses image with the buttons on the SD2IEC device this change will be done automatically. The swap file can be mounted without user interaction using the “XS:AUTOSWAP.LST” command. (See drive *firmware* documentations) Attention: in case of a disk image change (or any directory change), all previously opened channels will be closed!

7 VCPU program execution error handling

VCPU program execution can terminate for several reasons beside “SYSCALL_EXIT_xxx” system calls. When a system call with an unknown code is made a “97,VCPU ERROR,101,xx” error will be set where xx is the code of the unknown system function. For other terminate reasons the result will be “97,VCPU ERROR,100,xx” where xx will be the same as the **INT** parameter of the “ZC” command. The bits of this BYTE mean different events:

- B7: drive *firmware* compilation error. The programmer / user should not encounter this one. It means the firmware is not properly compiled.
- B6: Address error. When the VCPU would jump to an unavailable address the execution will be terminated with this signal.
- B5: Address error. The system call was made from an invalid address.
- B4: Invalid instruction. Execution “illegal” opcodes will be terminated with this signal.
- B3: SD card has been removed (or inserted if there was no card inserted previously).
- B2: Program execution is terminated due to ATN active. This can be disabled with the proper system calls!
- B1: Read / write address error. When the VCPU would read / write outside of the usable memory / I/O buffers the execution will be terminated with this signal.
- B0: Program execution is terminated due to system call. This will be set for exit system calls.

8 Remarks

8.1 1 bit / 2 bits

A lot of users does not use the SD2IEC drive as a single dive, they have a 15x1 drive connected too. The basic operation of the serial bus that in response to the computer's ATN active signal the peripherals pull the DAT line to low. Because of this the programs that use ATN for data transfer can be used only when there is only one peripheral on the serial bus! For these cases the SD2IEC drives provide a feature to "disconnect" them from the serial bus without actually plugging them off. It's important to note that in case of the original 15x1 drives this is not possible! (Turning off the drive is not enough, the drives must be plugged off!) Due to this behavior the SD2IEC transfer routines should be implemented to use ATN in very justified cases! Of the above, *synchronous transfer*s the 1 bit versions does not use ATN. Preferred communication protocols use CLK / DAT only!

8.2 Buffers / memory

The VCPU program memory is part of the drive *firmware* data buffers. For this reasons the programmer must be careful to not putting program code to the buffers used by file operations because they will be overwritten. The programmer (currently) can't control which operation will use which buffer but – fortunately – the buffer allocation / freeing is reliable. The drive *firmware* is allocating buffers from the beginning, in order but there are some thing to be considered. When the user (or the program itself) uses disk images the drive will allocate a buffer to handle that "internally". In generic uses cases this will be buffer 0. This buffer remains allocated after the user "exits" that disk image! In addition the opened file allocates a buffer too which in this case will be buffer 1. But if this file is not inside a disk image and the user haven't used a disk image yet this will be buffer 0! For this reason the number of the buffer used should be handled dynamically.

These two buffers will be the VCPU memory range `$0000..$00FF` and `$0100..$01FF`. Because of this the zero page and the stack must be relocatable and the VCPU supports this. The remaining memory for the drives with few buffers is scarce, 1 KBYTE which is half of the memory of the 1541. But the disk handling functions are provided by the drive *firmware* so the complex (and thus memory intensive) program codes are not required here.

Number of buffers is determined by the amount of RAM in the SD2IEC drive's microcontroller. As this was not really relevant before most of the users have an SD2IEC drive which allows 6

buffers only. Because of this the programmer should be prepared for this number. When more memory is required the users won't have the hardware to support that so they won't be able to run that program.

8.3 "Command channel" and "error channel" as memory

The "command channel" and "error channel" can be used to store temporary data while the VCPU program is running, but there is no guarantee that its contents will be preserved during system calls! (The same applies to the characters of the command in the "command channel"! Therefore, before calling any command, the whole command must be copied into the "command-channel", it is not sufficient to replace only the possibly modified characters.) It is not possible to run programs with VCPU from these memory areas!

8.4 Disk images / Directories

The different disk images can be accessed as simple directories with the drive *firmware*. Changing directories are done with the "`CD:...`" command. (See drive *firmware* documentation for more details.) In addition switching between directories / disk images is also possible using "`AUTOSWAP.LST`" (or any other, manually specified) file. (In normal operation stepping between the disk images / directories in the list is done by using the device buttons. During a VCPU program execution the `SYSCALL_CHANGEDISK` (`$31`) function is also possible.) Important: during **directory / disk image change all the open file / communication channel gets closed!** When a communication channel was open for the sector level handling of a disk image it must be reopened after the disk swap!

8.5 Storage speed

Although most of the load (save) time is made of the communication between drive ↔ computer but time for accessing the SD card and handling its file system is not negligible. But these times depend on the type of the card, the parameters of the file system (FAT type, cluster size, etc.), the location of the file, etc. The programmer must not assume that the times experienced during development will be met by the configuration of every user!

8.6 Filenames

Filenames on CBM drives can be up to 16 characters long. The file "extension" is not part of the name, and there are only a few variations. (`PRG`, `SEQ`, `USR`, `REL`.) *FlexSD fw* handles different

versions of the FAT file system on the SD card (FAT-12, 16 and 32). In this file system, file names were originally limited to 8 characters in length, with an additional 3 character “extension”, which in this case could be anything. The drive *firmware* uses the LFN extension of the FAT file system, which allows longer file names (up to 255 characters). However, for compatibility reasons, all files also have “short” names of size 8+3. If a FAT “long” filename is 16 characters or less, it is available with the same CBM filename. If the file name is longer than 16 characters, the CBM filename will be the 8+3 character short name of the FAT file! The length of the file name includes the extension characters, but the firmware allows extensions known from CBM drives not to appear at the end of the name. (See the `XE+`/`XE−` commands in the documentation.) Hiding extensions is not enabled by default (so extensions are visible)! When choosing a file name, it is therefore advisable to choose from the following:

- The filename should be up to 16 characters and should not have an extension
- The filename is up to 12 characters, and can be used with one of the usual CBM extensions

In the latter case, the filename should be specified in the program as “`FILENAME*`” instead of “`FILENAME.PRG`” or “`FILENAME`”, so that it does not matter whether the user has enabled hiding of known file extensions. For filenames with lengths of 13 to 16 characters, the visible extension may exceed 16 characters. Then the CBM name will be the FAT 8+3 character “short” filename! However, for hidden extensions, the “normal” filename will be the CBM name. Because of this behaviour, it is advisable to avoid filenames longer than 12 characters (or extensions written at the end of a FAT filename).

8.7 VCPU extended / “micro” instruction codes

Due to the few resources of the hardware of the most popular SD2IEC drives compatibility with an old CBM drive is not a goal. Because of this 100% compatibility of the CPU emulation with the original 6502 is also not required. Support for the “undocumented” instructions is missing for this very reason but the usability of those instructions is also limited. Because it does not need to be compatible with an existing hardware it would be useful to implement the additional instructions / addressing modes of the 65**C**02 CPU instead of the “undocumented” instructions. The codes of the VCPU extra instructions has been selected to occupy the places of the unused instructions of the 65C02 CPU so later this processors new instructions / addressing modes can be implemented! (Currently there is not enough free place for devices with smaller memory.)

8.8 Synchronisation

As described in the "ZE" command, the time between issuing the run command via KERNAL and starting the VCPU program may vary, depending on the type of SD2IEC drive / *firmware*! For this reason, it is useful to sign to the computer that the VCPU program execution has started. The simplest method is for the drive to set the CLK line to low, then wait for the computer to detect this and acknowledge it with a low pulse on the DAT line. The VCPU program can then start like this:

```
UCLDH           ;CLK line low, DAT line release
UWDTL           ;Wait until the DAT line is low
USCKH           ;CLK line release
UWDTH           ;Wait until the DAT line is high
```

After issuing the "ZE" command, the computer program will wait until the CLK line is low. As soon as this happens, switch DAT to low and then release it. The signaling is done by the drive with the CLK line, and it is not recommended to do it with the DAT line!

(The machine will release the ATN line at the end of the "ZE" command, then the CLK line, and the drive will release the DAT in response. But KERNAL does not wait for the DAT high state! Since the DAT line may remain low for an "unspecified" period of time after the command is issued, the synchronization routine may detect it as low (instead of starting the VCPU program). This "misinterpretation" does not occur when using the CLK line.)

8.9 "Bus type"

The value of the "bus type", which can be queried using the "ZI" command or read from the I/O registers, rarely needs to be checked. However, if this is necessary, you should pay attention to the following:

- If CBM SERIAL ($2) is required to run the program, CBM FAST SERIAL ($3), CBM SERIAL + PARALLEL ($4) or CBM FAST SERIAL + PARALLEL ($5) must be accepted;
- If CBM FAST SERIAL ($3) is required to run the program, CBM FAST SERIAL + PARALLEL ($5) must be accepted;
- If CBM SERIAL + PARALLEL ($4) is required to run the program, CBM FAST SERIAL + PARALLEL ($5) must be accepted;

The VCPU version code should always be handled separately! The version code does not depend on the bus type(s) supported by the actual drive.

8.10 “Fast Serial” bus

This data transfer is only supported by the C128 at the factory! If the drive firmware includes support, this is indicated by the version number of the VCPU (see DOS command **“ZI”** or I/O register description). However, *firmware* support does not mean that the device can be used for “Fast Serial” data transfer! A small number of SD2IEC drives have dropped the SRQ line handling for simplification/cost reasons, so it is advisable to test the data transfer before use. This is done by the drive sending a BYTE to the computer (`FSTXB` command), which is checked by the receiving side. (If the drive can send data, the other direction is probably working.)

When transmitting data, the receiving side must be synchronous with the sending side (the bit being transmitted must be in the same position on the receiving side as it was on the sending side.) On the drive side, the data transmission (`FSTXB` instruction) always initializes the data reception! (8 data bits must always be received after the BYTE sent.) This initialization is also performed by the data reception enable (`FSREN` command), regardless of whether the reception was previously enabled! For this reason, if this instruction is executed while data reception is in progress, the communication will be lost. However, a previous out-of-sync state can be restored by doing this. (Cyclical disabling/enabling of reception is usually not necessary, enabling it once at the beginning of the program, after which the data transmission will automatically handle it.)

Attention: if the bus type returned in the response to the "ZI" DOS command is CBM FAST SERIAL, this only means that supported by the VCPU! Working with the computer (C128) KERNAL is independent of this, it is not necessarily implemented! (In the current firmware, KERNAL communication is only available in devices with larger program memory!)

9 About the sample sources

9.1 The “#”

The VCPU has several new instructions in addition to the original 6502. These instructions mostly use *immediate* addressing mode. This addressing mode is marked by “#” in the original 6502 *assembler* syntax (e.g. `#$55`), in the previous parts of the documentation these instructions are also presented this way. Currently there is no *assembler* supporting VCPU extended instructions, so a macro definitions source has been created to support using them. However the *assemblers’* macro definitions usually can’t process a parameter which is *number* (or a label

representing it) and begins with the symbol “#”! For this reason these instructions, defined as macros the “#” symbol is left out from the parameters. So they are written as

`BREAK $00` instead of `BREAK #$00`

form. But this does not mean that the parameter of these instructions is a (zeropage) address! As all of the “new” instructions has only one addressing mode this won’t cause a misunderstanding. If in the future there will be an *assembler* supporting these instructions directly then the original syntax using “#” should be implemented but for compatibility reasons the one without the “#” can be accepted too.

9.2 Supported architectures for the sample programs

The sample programs can be compiled for 4 different architectures. These are VIC20, C64, C16 / C116 / plus/4 (C264 series) and C128. The KERNAL level programming of these machines are the same but there are programs which demonstrate the data transfer between the drive and the computer and these are hardware dependent. The purpose of the samples is rather demonstrate the SD2IEC + VCPU than the programming of the computer so – to keep it simple – the VCPU sources are the same for all platforms. This also means that a sample showcasing the data transfer is not optimal for all platforms! (For example the computer hardware would require different bit order for every platform). (Ones of) The goal(s) of these samples is to aid the programmer to implement the optimal data transfer for the chosen platform.

9.3 License for the sample programs

The sample programs are made because they can be helpful. Knowing the **“there is no guarantee for anything, not even basic operations”** remark they can be used, wholly or partially freely, without restrictions.

9.4 Description of the sample programs

The sample programs are made to test the VCPU extension functions but they can be useful to answer questions about programming the device. All of the programs begin with checking the peripherals of the computer, counting the number of them connected to the serial bus. Then it searches the SD2IEC drive (when there are more it chooses the last one) and does its tasks with this drive. The functions of these programs are the following:

- A-DETECT: Prints the long version ("X?") of the found drive. Check for VCPU support ("ZI") and displays the response. After that it prints the number of buffer ("ZB") and the current state of the VCPU ("ZC"). It also includes a simple drive configuration utility.
- B-SHOWMEM: Displays the memory handled by the SD2IEC VCPU extension ("ZR").
- C-PRINTIO: Displays the I/O area visible for the VCPU. This (and the rest of the samples) requires a program to be downloaded to the drive.
- D-BUTTLED: The two LEDs of the SD2IEC ("BUSY" / "DIRTY") can be controlled by the two buttons ("PREV" / "NEXT"). In addition the computer's CLK / DAT lines also control these LEDs (**C** + **V** and **D** + **F** keys). At exit it queries the drive status which requires ATN activity causing the VCPU to interrupt execution. Because of this the status will display the matching error code. After this the VCPU state is also displayed.
- E-SR1B-PIO: The computer send 256 BYTEs to the SD2IEC then the drive sends this packet back to the computer. It checks if it's the same as it was sent. If they match the sending begins anew. This test runs until the user exits or there is a comparison mismatch. The received data is displayed on the screen. This test uses CLK / DAT lines only. DAT is used for transferring data bits and CLK is used by the computer for timing the transfer. The VCPU side of the communication is implemented using the I/O registers in the "old fashioned" way. This transfer mode is not recommended!
- F-SR1B-USR: Data transfer test. The transferred data is the same as the one described for E-SR1B-PIO. This one also uses CLK / DAT only. The VCPU side implementation uses USND1 / URCV1 instructions. This is a recommended method!
- G-SR1B-URE: Data transfer test. The transferred data is the same as the one described for E-SR1B-PIO. This one also uses CLK / DAT only. The VCPU side implementation uses UDTTA / UATDT instructions. This is a recommended method! The order of the bits on the DAT line is reversed compared to the previous transfer protocols.
- H-RECVTIME1B: Data receive test, receives and checks a specified number of BYTEs for every frame. Uses CLK / DAT lines only.
- I-SR2B-PIO: Data transfer test, the two bit equivalent of E-SR1B-PIO. In one step 2 bits are moved on the CLK / DAT lines and the ATN is used for timing. For two bit transfers using ATN only one drive (in this case the SD2IEC) can be connected to the serial bus of the computer! The VCPU side of the communication is implemented using the I/O registers. This transfer mode is not recommended!
- J-SR2B-USR: Data transfer test, the two bit equivalent of F-SR1B-USR. The data transfer is the same as described for I-SR2B-PIO. The VCPU side implementation uses USND2 / URCV2 instructions. This is a recommended method!
- K-SR2B-URE: Data transfer test, the two bit equivalent of G-SR1B-URE. The VCPU side implementation uses UCDTA / UATCD instructions. This is a recommended method! The order of the bits on the DAT line is reversed compared to the previous transfer protocols.
- L-RECVTIME2B: Data receive test, the two bit equivalent of H-RECVTIME1B.
- M-LOADTST1B: File load test. 1 bit data transfer using CLK / DAT lines. There must be an SD card in the drive with the test file "TSTDAT2M.SEQ" in the current directory. This is a 2 MBYTEs file consisting of random numbers. The test reads this file four times. The difference between reads are block size (254 or 256 BYTEs) and the presence of checksum (calculated or skipped). The process measures "time" for reading. (The "time" is the number of interrupts for the used platform which is ~60 for VIC20 / C64, for (PAL) C16 / C116 /

plus/4 and (PAL) C128 is ~50 per second. If the counted value is divided by this number the load time can be calculated. The values directly not applicable to compare platforms!

- N-LOADTST2B: File load test. The two bit equivalent of M-LOADTST1B using CLK / DAT and ATN lines.
- O-DISKIMGS: Disk image handling test. There must be an SD card in the drive with the "VCPUTSTDSK1.D64" and "VCPUTSTDSK2.D64" disk images in the current directory. The test enters the first then the second disk images using "CD:..." command. Then reads the 3-3 files in them with direct sector read commands. It calculates a checksum of the files which is (among other data) is returned to the computer which displays / checks them.
- P-AUTOSWAP: Disk image handling test. The tests described for O-DISKIMGS will be processed but the disk change is performed using the "AUTOSWAP.LST" file. (This one is also required in the current directory of the SD card besides the disk images).
- Q-BENCHMARK: File read speed test. The VCPU test program reads the "TSTDAT2M.SEQ" test file used by the load tests using drive *firmware* and the computer measures the read "time". It repeats this with both 254 and 256 BYTEs block size. This test is used for testing the "raw" file read performance of the drive! The measured "time" is calculated in the same way as for file load tests.
- R-LSTRESSTST: From the SD2IEC side, a "stress test" of the serial lines. It can be used to detect certain types of hardware faults.
- S-40TRKTST: 40-track .D64 disk image test. To execute, the image "VCPUTST40TRK.D64" and the file "AUTOSWAP.LST" are required in the current directory. The test reads through all sectors of the mounted image and then checks/writes the values for the track + sector + logical block number. It then opens it as a file and checks that the data entered is in the correct place.
- T-SRFS: Data transfer test, Fast Serial equivalent of F-SR1B-USR, C128 only!
- U-RECVTIMEFS: Data receive test, Fast Serial equivalent of H-RECVTIME1B, C128 only!
- V-LOADTSTFS: File load test, Fast Serial equivalent of M-LOADTST1B, C128 only!
- 1-SRPP: Data transfer test, parallel port equivalent of E-SR1B-USR.
- 2-RECVTIMEPP: Data receive test, parallel port equivalent of H-RECVTIME1B.
- 3-LOADTSTPP: File load test, parallel port equivalent of M-LOADTST1B.
- Y-LINEDIAG: Test of the serial lines' state from the SD2IEC side. Diagnostics program, may be removed in the future!
- Z-VCPUCTST: Test for the VCPU core. Checks for running the emulated 6502 instructions. Not a full, "comprehensive" test! Currently all (original 6502 and extended) instructions are run / tested with at least one addressing mode. ("Micro" instructions are not tested here, most of them are tested in the data transfer tests.) The occasional VCPU instruction errors may get a dedicated test in this program together with their fixes.

10 Known bugs

- VCPU program execution allows only from RAM area! But address of the instruction to be executed is checked only for unconditional jumps. This can cause that during a simple program execution the **PC** can run out of the area designated as RAM. Additionally the conditional jumps can lead to jumping out of allowed memory range. In these cases the operation is “undefined”. Write to an invalid memory address is not executed (just like in case of normal program execution) so this does not affect normal operation of the drive *firmware*. System function calls (`$00: BREAK` code) from these invalid addresses are also not executed.
- The traditional interrupt handling is missing! Instead of that there are several events that interrupt VCPU program execution (such event is for example removal of the SD Card). These events are not checked after execution of every instruction just after the ones that allow creating program loops. (These are typically the jump / wait “micro” instructions.) Because of this the program is not interrupted when the event is happening but after one of the following jump instructions! In these cases the VCPU status query value of **PC** won’t reflect the actual instruction when the event happened.
- Execution time of instructions accessing the I/O area depends on the address of the register. Accessing the same register mostly takes the same time, independently of the written / read data.
- The binary → decimal converter peripheral converts binary values to decimal / ASCII digits using simple cycles so execution time (only for write this I/O registers) depends on the value to be converted!
- Execution times for the original 6502 instructions are not defined (with several exceptions). Possible optimizations can change them in the future!
- The data transmission of the “Fast Serial” bus is implemented in software. During data reception, ATN state changes may cause unexpected behaviour (missed bits in received data, ATN change may go unnoticed). For this reason, the computer side should avoid switching the ATN while the data transfer is in progress (even on 1571 / 1581 drives, data will be corrupted in such a case, this is a CBM SERIAL bus specificity.)

Table of contents

Change log:
221002: First public release
230326: This→Those, Introduction→Features list, Extend VCPU bus-types, `OPEN` / `$FF`, Channels+memory chapter, Filenames chapter, `ZI` extension, remove BTASC instruction, Binary→decimal converter peripheral, I/O register table + extension, `PHA`+`PLA`, documentation version corresponds to the firmware version
230404: Spelling correction
230610: Spelling correction, `R-LSTRESSTST` added
230616: System calls / `SYSCALL_OPEN` error handling clarification, `SYSCALL_CHANGEDISK` channel close
231010: VCPU R2 addendum (preliminary), chapter 8.8, `S-40TRKTST`
240122: VCPU R2 Fast Serial addendum (preliminary), `T-SRFS`, `U-RECVTIMEFS`, `V-LOADTSTFS`, OPEN addendum, corrections
240925: R2 "preliminary" comment removed, Parallel port addition, Fast Serial KERNAL comment, other clarifications