



# Nascom2 for the MEGA65

## Documentation for the MEGA65 Core

# DOCUMENTATION

Generated from Markdown source

*This documentation describes Nascom2 for the MEGA65.  
Specifications may change with future core releases.  
Prepared in a style reminiscent of early 1980s microcomputer manuals.*

Thomas Linke  
April - September 2026  
NASCOM / MEGA65 Core Documentation

**MEGA65**

## CONTENTS

|                                                   |    |
|---------------------------------------------------|----|
| 1. CHANGELOG                                      | 6  |
| 2. WHAT THE NASCOM2 EMULATOR CAN CURRENTLY DO     | 6  |
| 3. F5 HARDWARE MONITOR                            | 8  |
| 4. F7 CP/M FDD DIAGNOSTICS                        | 9  |
| 5. RESOLUTION                                     | 9  |
| 6. MEMORY MAP                                     | 9  |
| 7. FIRST STEPS                                    | 10 |
| 8. FIRST STEPS WITH FDD                           | 10 |
| 8-1 List of all FDD commands . . . . .            | 10 |
| 9. FIRST STEPS WITH AY SOUND                      | 11 |
| 9-1 Trying all three tone channels . . . . .      | 11 |
| 9-2 Trying the noise generator . . . . .          | 12 |
| 9-3 Trying the envelope generator . . . . .       | 12 |
| 10. FIRST STEPS WITH THE JOYSTICKS                | 12 |
| 11. CP/M 2.2                                      | 12 |
| 11-1 Commands & Tools on Disk CPM22.DSK . . . . . | 13 |
| 11-2 Network Disk CPMNET.DSK . . . . .            | 14 |
| 12. FIRST STEPS WITH AVC                          | 14 |
| 12-1 Enabling the AVC . . . . .                   | 14 |
| 12-2 Trying the AVC under CP/M . . . . .          | 15 |
| 12-3 Using G48 with ROM BASIC . . . . .           | 15 |
| 13. NAS-DOS ROM PATCH                             | 16 |
| 14. REAL-TIME CLOCK SUPPORT                       | 18 |
| 15. PMOD RS232 UART COMMUNICATION                 | 18 |
| 15-1 PMOD RS232 Pinout . . . . .                  | 18 |
| 16. ETHERNET AND NETWORK PROGRAMS                 | 19 |
| 16-1 Network configuration . . . . .              | 19 |

|                                                                    |    |
|--------------------------------------------------------------------|----|
| 16-2 PING.NAS . . . . .                                            | 19 |
| 16-3 TELNET.NAS . . . . .                                          | 20 |
| 16-4 FTP.NAS . . . . .                                             | 20 |
| 16-5 CONTROL.NAS . . . . .                                         | 20 |
| 16-6 Notes and troubleshooting . . . . .                           | 21 |
| 17. CONFIGURATION FILE N2CONF.CFG                                  | 21 |
| 17-1 Current Format . . . . .                                      | 21 |
| 17-2 Byte Layout . . . . .                                         | 21 |
| 17-3 Bit Layout . . . . .                                          | 22 |
| 17-4 Byte 6 = cfg_byte4 . . . . .                                  | 22 |
| 17-5 Byte 7 = cfg_byte5 . . . . .                                  | 22 |
| 17-6 Byte 8 = cfg_byte6 . . . . .                                  | 23 |
| 17-7 Byte 9 = cfg_byte8 . . . . .                                  | 23 |
| 17-8 Encoded Values . . . . .                                      | 23 |
| 17-9 Checksum . . . . .                                            | 24 |
| 17-10 Values Read Back . . . . .                                   | 24 |
| 17-11 From Byte 6 . . . . .                                        | 24 |
| 17-12 From Byte 7 . . . . .                                        | 24 |
| 17-13 From Byte 8 . . . . .                                        | 25 |
| 17-14 From Byte 9 . . . . .                                        | 25 |
| 17-15 Backward Compatibility . . . . .                             | 25 |
| 18. Z80 I/O PORTS                                                  | 26 |
| 18-1 Port Changes from the Original CP/M Design . . . . .          | 26 |
| 18-2 NASCOM Basic Functions . . . . .                              | 26 |
| 18-3 AY-3-8910 Programmable Sound Generator . . . . .              | 27 |
| 18-4 Lucas Logic Advanced Video Controller (AVC Model B) . . . . . | 28 |
| 18-5 Original G48 ROM BASIC graphics software . . . . .            | 29 |
| 18-6 Real-Time Clock . . . . .                                     | 29 |
| 18-7 WD1793/Floppy Controller . . . . .                            | 30 |
| 18-8 CP/M/FDC Diagnostic Ports . . . . .                           | 31 |

|                                                            |    |
|------------------------------------------------------------|----|
| 19. SOURCES AND ACKNOWLEDGEMENTS                           | 31 |
| 20. LICENSING                                              | 31 |
| 21. AUTHOR                                                 | 32 |
| ADDENDUM 01                                                | 33 |
| NASSYS Ping client . . . . .                               | 34 |
| NASSYS Telnet client . . . . .                             | 34 |
| NASSYS FTP client . . . . .                                | 34 |
| ADDENDUM 02                                                | 35 |
| Binary framing . . . . .                                   | 35 |
| ADDENDUM 03                                                | 36 |
| ADDENDUM 04                                                | 37 |
| ADDENDUM 05                                                | 38 |
| Loading and starting . . . . .                             | 38 |
| Login . . . . .                                            | 38 |
| Commands . . . . .                                         | 38 |
| Transport . . . . .                                        | 39 |
| Firewall and passive data ports . . . . .                  | 39 |
| Local test server . . . . .                                | 39 |
| ADDENDUM 06                                                | 40 |
| 1. Starten . . . . .                                       | 40 |
| 2. Das erste Programm . . . . .                            | 40 |
| 3. Beispiel mit Eingabe und Variable . . . . .             | 41 |
| 4. Die wichtigsten Befehle am >-Prompt . . . . .           | 41 |
| 5. Grundlegende Editorbedienung . . . . .                  | 41 |
| 6. Compiler- und Laufzeitfehler . . . . .                  | 42 |
| 7. Speicherhinweis für unsere \$A000-ROM-Version . . . . . | 42 |
| 8. Nützliche Sprachelemente für den Anfang . . . . .       | 42 |
| Quellen und Versionshinweis . . . . .                      | 43 |
| ADDENDUM 07                                                | 44 |

|                                           |    |
|-------------------------------------------|----|
| Overview . . . . .                        | 44 |
| Recommended connection sequence . . . . . | 44 |
| TCP is a byte stream . . . . .            | 45 |
| Request frame . . . . .                   | 45 |
| Response frame . . . . .                  | 45 |
| XOR checksum . . . . .                    | 45 |
| Sequence number . . . . .                 | 46 |
| Commands . . . . .                        | 46 |
| PING (01h) . . . . .                      | 46 |
| Read memory (10h) . . . . .               | 46 |
| Write memory (11h) . . . . .              | 47 |
| Read port (20h) . . . . .                 | 47 |
| Write port (21h) . . . . .                | 47 |
| Execute Z80 code (30h) . . . . .          | 47 |
| End session (7Fh) . . . . .               | 47 |
| Status codes . . . . .                    | 47 |
| Robust response parser . . . . .          | 47 |
| Block transfers . . . . .                 | 48 |
| GUI recommendations . . . . .             | 48 |
| Reference implementation . . . . .        | 49 |

When I decided to get into FPGA programming and vintage computers, I had no idea where it would lead.

I thought I would simply create a Z80EMUF, a single-board computer with a Z80, some RAM, ROM, and I/O.

When I realized how straightforward that was, I wanted to continue. Since the Nascom2 is one of my favorite computers, I carried on with that - and I still enjoy the result every time I use it. I am also not running out of ideas for extensions yet.

## **1. CHANGELOG**

- 2026-04-26 - local V0.23 / Mega65-Filehost V1.0 - initial public version
- 2026-06-09 - V1.1 - pictures and manual
- 2026-06-23 - V1.2 - PMOD-RS232-Support, VGA-Output and Scanlines
- 2026-07-15 - V1.3 - PowerLED in orange, I2C RTC Support from Mega65R6
- 2026-07-22 - V1.4 - Z80 Hardware Monitor via F5
- 2026-07-30 - V1.5 - SingleStep and Exec for Debug Mode, VideoZoom, FDD-Support
- 2026-08-08 - V1.6 - CP/M 2.2 und AVC Graphic Card, changed native screen resolution 800x600@60Hz, new software
- 2026-08-18 - V1.7 - R3-Compatibility is now fixed and tested, added an IP-Stack for Ethernet, RENAME and DELETE is possible in SD-Browser
- 2026-08-19 - V1.74 / Filehost V1.8 - fixed amber/green phosphor colours in the AVC 96-column terminal and monochrome double-density mode;  
the SD browser now identifies .DSK images as [CP/M] or [NASSYS]
- 2026-09-03 - V1.9 - added an AY-3-8910-compatible three-channel sound generator with 48 kHz 16-bit PCM output over HDMI and 3.5 mm jack  
(AK4432VT DAC on R6, one-bit audio on R3/R3A), HDMI and VGA-path fixed, Joystick-Support (PIO from AY-3-8910), now there is a  
website to to this project: <<https://nascom2.de>>
- Runs on the MEGA65, tested on R3A (thanks to Jerrec) and R6, and also built for R5.  
Feedback on whether it runs correctly is welcome!

## **2. WHAT THE NASCOM2 EMULATOR CAN CURRENTLY DO**

- T80s emulation
- By default, the Nascom-Memory is configured like my own machine. See Memory Map.
- It is partially configurable:
- BASIC and NAS-SYS3 are enabled in the normal memory map
- The ROM area from \$B000 to \$DFFF (\$CFFF, if FDD is active) can be switched between RAM and ROM
- BLS PASCAL V1.3 on configuration page 2 maps the 12 KiB Pascal ROM at \$A000-\$CFFF. BASIC and the other expansion ROMs are hidden while this mode is active; \$E000-\$FFFF becomes writable RAM.
- Helpful overlay screens are available via function keys:
- F1 - MEGA65 to Nascom2 keyboard mapping
- F3 - Overview of NAS-SYS3 commands
- F5 - Hardware monitor (CPU state, RAM editor, single step, and execute)
- F7 - CP/M FDD Diagnostics
- F9 - Save file
- F11 - File browser for loading files
- F13 - Configuration and reset

- The CPU is paused while an overlay is displayed
- SD card support
- Files in .NAS, .CAS, and headerless 640 KB .DSK format can be read; mounted .DSK images can also be updated by NAS-DOS
- Enable FDD / NAS-DOS on the second configuration page to map NAS-DOS at \$D000-\$DFFF and the Lucas/WD1793-compatible controller at ports \$E0-\$E5
- CP/M-MODUS appears directly below it. Enabling it forces FDD on, forces all four configurable ROM sockets to RAM, and keeps \$D000-\$DFFF writable instead of mapping NAS-DOS there. The previous ROM/RAM socket selection is retained and restored when CP/M mode is disabled again.
- CP/M mode boots the existing CP/M 2.2 BOOT rev 2.1 2 KiB EPROM. At reset it is visible at \$F000-\$F7FF and mirrored at \$0000-\$07FF for the reset vector. The generated FPGA image changes its bank-port operand from the historical \$B2 to private port \$BF, because \$B2 belongs to the AVC. OUT (\$BF), \$80 switches \$0000-\$0BFF to shadow RAM while the boot overlay retains relocated video RAM at \$F800-\$FBFF. Bit 6 at port \$BF is reserved for the new second-stage loader: \$C0 exposes the high RAM shadow at \$F000-\$FFFF.
- AVC appears below CP/M-MODUS. It enables or completely disconnects the Lucas Logic AVC emulation. When disabled, ports \$B0-\$B2 and its \$8000-\$BFFF plane window are inactive and normal NASCOM video is selected.
- In the file browser, ENTER toggles a selected .DSK for logical disks 0/1 and SHIFT+ENTER toggles it for disks 2/3. Mounts are shown beside each image. .DSK files are hidden while FDD / NAS-DOS is disabled; enabling CP/M also enables FDD and makes them visible again.
- Press R to rename the selected file's 1-8 character base name. The existing three-character extension is retained; press ENTER, then Y to confirm.
- Press Delete, then Y, to delete the selected file. Deletion also releases the complete FAT cluster chain in both FAT copies. N or Esc cancels either confirmation.
- Rename/delete deliberately supports plain FAT 8.3 entries only. Files with a long-filename entry, read-only files, and currently mounted .DSK images are protected. Unmount a DSK before changing or deleting it.
- DSK sector access is read/write. Images use 80 tracks, two sides, 16 sectors per track, and 256 bytes per sector. WD1793 WRITE TRACK is supported for NAS-DOS formatting.
- Formatting is non-destructive: on the first WRITE TRACK, the mounted NAME.DSK is renamed to NAME.BAK, a fresh 640 KB NAME.DSK is allocated, and subsequent track data is written only to the new image. If NAME.BAK already exists, formatting stops with a write fault instead of overwriting it.
- All files are read from and written to a /NASCOM2 directory on the SD card. Various cards have been tested, up to 128 GB.
- Files are saved in .NAS format. Memory ranges and the filename can be entered.
- If BASIC has been initialized, the system checks whether a BASIC program exists. If it is to be saved, the addresses are read automatically.

- The motherboard LEDs (TAPE and HALT) are shown at the edge of the screen
- The single-step NMI path is supported in software. Shift+CLR/Home operates the manual NMI button; Ctrl+CLR/Home is supported as an alternative. Plain CLR/Home is deliberately harmless so an accidental press cannot destroy a BASIC program. Under CP/M the chord performs a safe warm boot back to the command prompt.
- A system reset can be triggered from the configuration screen, as well as with the reset button on the device
- The CPU speed can be switched between 1, 2, and 4 MHz
- The screen colors can be switched between amber and green, now with scanline-support. The selected phosphor colour is also used by the AVC 96-column CP/M terminal and monochrome double-density graphics.
- Settings are saved on the SD card
- UART RX/TX are supported via Mega65 USB JTAG-Adapter (115200 8N1)
- With an PMODRS232-Adapter a real UART RS232-Connection with RTS/CTS-Support is possible. Baud and RTS/CTS are switchable.
- Power-LED in nice orange and Drive-LED is active if WD1793-FDD is working
- I2C real-time clock support on R3/R3A and R6. The core automatically prefers a valid external DS3231 on the Grove connector and otherwise uses the onboard RTC.
- AY-3-8910-compatible three-channel sound, noise, and envelope generation over HDMI audio and the MEGA65 3.5 mm audio jack (AK4432VT DAC on R6; one-bit output on R3/R3A)
- Both physical MEGA65 joystick ports are available through the AY-3-8910 parallel I/O ports. Digital directions and the fire button are supported on R3/R3A and R6; analogue paddles are not currently implemented.
- Ethernet support on MEGA65 R3/R3A and R6 provides static IPv4, ARP, ICMP ping, DNS, and outbound TCP connections.

### **3. F5 HARDWARE MONITOR**

F5 opens a hardware-level monitor and pauses the emulated CPU. The monitor shows AF, BC, DE, HL, their alternate register set, IX, IY, SP, PC, I, R, the current instruction register, interrupt mode/IFF state, machine cycle, T-state, and all flags.

The lower half of the screen is a 64-byte memory viewer and RAM editor:

- Use the cursor keys to select a RAM byte; hexadecimal keys edit its value.
- Press cursor up in the first RAM row to enter an address directly.
- During direct address entry, type four hexadecimal digits (for example A000) and press Enter. Cursor Left/Right selects a digit; Escape or Cursor Down cancels the entry.
- - / + move backward or forward by 64 bytes. Page Up / Page Down work as aliases when an external PC keyboard provides them.
- Home opens the memory page containing PC.
- ROM and optional ROM-slot contents can be inspected but are read-only. RAM bytes can be overwritten by entering two hexadecimal digits (0-9 or

A-F); the cursor then advances automatically.

- SPACE executes one Z80 instruction and returns to the paused monitor.
- ENTER loads the selected RAM address into the program counter and keeps the CPU paused in the hardware monitor. Continue from that address one instruction at a time with SPACE. This is the hardware-monitor equivalent of selecting a start address with the NAS-SYS E command and does not require NAS-SYS to be active.
- Since F5 pauses the CPU immediately, the first single step may finish an instruction that was already in progress. Every following step executes exactly one complete instruction.
- F5 closes the monitor and resumes the CPU.

#### **4. F7 CP/M FDD DIAGNOSTICS**

The CP/M FDD Diagnostics overlay is actually a by-product of the CP/M development work. However, since it shows a lot of interesting information, I will leave it in the core for now.

#### **5. RESOLUTION**

- Screen: 48 x 16 characters
- Font: 8 x 16 pixels
- Total: 384 x 256 pixels

Video output is via HDMI and VGA at 800 x 600 @ 60 Hz.

Pixel scaling was deliberately avoided, since in 1980 people also worked with 9- to 12-inch monitors. On a 24- or 27-inch TFT, this gives a good approximation of the size of the screens used at the time.

The configuration menu offers ORIGINAL, ZOOM 150%, and ZOOM 200% while keeping the 800 x 600 @ 60 Hz output timing and the complete logical 384 x 256 Nascom screen. The three displayed sizes are 384 x 256, 576 x 384, and 768 x 512 pixels. At 200% only 16 horizontal and 44 vertical border pixels remain on each side. ORIGINAL remains the default.

#### **6. MEMORY MAP**

Nascom2 base configuration:

| Address Range           | Type | Size  | Description                                 |
|-------------------------|------|-------|---------------------------------------------|
| ----- ----- ----- ----- |      |       |                                             |
| \$0000-\$07FF           | ROM  | 2 KB  | NAS-SYS3 Monitor                            |
| \$0800-\$0BFF           | RAM  | 1 KB  | Video Display Memory                        |
| \$0C00-\$0FFF           | RAM  | 1 KB  | Workspace used by various standard programs |
| \$1000-\$AFFF           | RAM  | 40 KB | RAM                                         |
| \$B000-\$B7FF           | ROM  | 2 KB  | Toolkit ROM (or RAM)                        |
| \$B800-\$BFFF           | ROM  | 2 KB  | NASPEN ROM (or RAM)                         |
| \$C000-\$C3FF           | ROM  | 1 KB  | SUPERDEBUG ROM (or RAM)                     |
| \$C400-\$CFFF           | ROM  | 3 KB  | NAS-DIS ROM (or RAM)                        |
| \$D000-\$DFFF           | ROM  | 4 KB  | ZEAP Assembler V2.1 ROM / NAS-DOS (or RAM)  |

| \$E000-\$FFFF | ROM | 8 KB | BASIC ROM |

## **7. FIRST STEPS**

- start the Nascom2 core on your MEGA65
- type T1000 and ENTER, you see the memory at address \$1000 - \$1007
- now type M1000 and ENTER, then type 23 ENTER, 42 ENTER, FF ENTER . ENTER
- now type T1000 again and you see the 1000 23 42 FF ... you changed your memory. Congratulations!
- now press F11 and with Cursor down go to GALAXIAN.NAS and press ENTER
- after a few time in the bottom line you can find the start address (here it is A1000)
- press F11 to go back to the NAS-SYS3-screen and type E1000 and ENTER
- play a round GALAXY ATTACK and have fun!
- if you broke the highscore, press F13 and ENTER (or press the MEGA65 reset button)
- type J and ENTER, you will be asked for 'Memory size?', press ENTER again
- now you are in the built-in Microsoft-Basic
- You can leave if you type MONITOR and ENTER or you can write a funny program in MS BASIC

## **8. FIRST STEPS WITH FDD**

The standard Nascom did not have a floppy drive, but later there were solutions using expansion cards. Lucas' solution used a WD1793 together with an additional EPROM, NAS-DOS.

This card is now supported as well. However, without an explanation, it is difficult to use.

USAGE: FDD support must be enabled via F11 by pressing it twice to reach the second configuration page. This automatically disables the ZEAP ROM/RAM at \$D000.

After that, .DSK files can be mounted from the file browser via F11. Press Enter to load the image into drive 1, sides 0/1. Press Shift+Enter to load it into drive 2, sides 2/3.

The browser reads the first sector of every .DSK image while scanning the directory. Images with the NASCOM CP/M NCB boot signature are marked

[CP/M]; all other supported disk images are marked [NASSYS].

Before anything can be done, the ROM must first be activated in NAS-SYS3 by entering E D000 and pressing Enter.

Afterwards, the directory of drive 1, side 1 can be displayed with ]D (it is similar to ]D0) followed by Enter. With ]D1, the second side of the first drive is shown. Similarly, ]D2 and ]D3 are used for the second drive.

All floppy disk commands start with ].

For example, a file can be loaded and executed directly with ]E:NAME: followed by Enter.

### **8-1 List of all FDD commands**

(d)=Disk-Number (0-3, 0 is default); all Options in () are optional;

]B(d)(:filename:) - Save Basic program

]C(d) - Clean Directory (Remove deleted entries)

]D(d) - Display the Directory

]E(d):filename: - Execute a program

]F(d):diskname: - Format disk

]L(d):filename: - Load program/file, but do not execute

```

]O(d):filename:llll hhhh eeee - Save object code program llll=startaddress; hhhh-1=end;
eeee=address of execution
]P(d)(:filename:) - Save NAS-PEN word processor file
]Rllll nnnn tt ss d - Read directly from disk nnnn sectors in Memory Location llll from
track tt, sector ss from drive d
]S(d):filename: - Scratch (delete) a file
]U(d) - Load user boot
]V(d):filename: - Verify that a file can be re-read
]Wllll hhhh tt ss d - Write directly to disk. Memory from llll to hhhh will be written to
track tt, sector ss on drive d. Be aware that there are no checks to determine whether it
is empty!
]Z(d)(:filename:) - Save ZEAP source file

```

## **9. FIRST STEPS WITH AY SOUND**

You need the AY-enabled bitstream and either an HDMI display or receiver capable of playing HDMI audio, or an active speaker/amplifier connected to the MEGA65 3.5 mm audio jack. Audio is sent to both left and right channels. There is no audio on the VGA output.

Start the built-in Microsoft BASIC from NAS-SYS by typing J and pressing ENTER. Press ENTER again at the Memory size? prompt. Then enter this first test program exactly as shown:

```

10 REM AY CHANNEL A - ABOUT 1 KHZ
20 OUT 8,0:OUT 9,125
30 OUT 8,1:OUT 9,0
40 OUT 8,7:OUT 9,62
50 OUT 8,8:OUT 9,15

```

Type RUN and press ENTER. You should hear a steady tone of approximately 1 kHz. The sound continues after the program ends because the AY registers retain their values. Stop it with this direct BASIC command:

```
OUT 8,8:OUT 9,0
```

If there is no sound, first check that the HDMI receiver or the device connected to the 3.5 mm jack is not muted and that the AY-enabled bitstream is running. For headphones, the MEGA65 headphone amplifier may also need to be enabled. The BASIC numbers above are decimal: port 8 selects an AY register and port 9 writes its value.

### **9-1 Trying all three tone channels**

The following program produces approximately 1 kHz on channel A, 500 Hz on channel B, and 250 Hz on channel C. The different volume levels make the three channels easier to distinguish:

```

10 FOR R=0 TO 15:OUT 8,R:OUT 9,0:NEXT R
20 DATA 0,125,1,0,2,250,3,0,4,244,5,1
30 DATA 7,56,8,15,9,12,10,10
40 FOR I=1 TO 10:READ R,V:OUT 8,R:OUT 9,V:NEXT I

```

Stop all three channels with:

```
FOR R=8 TO 10:OUT 8,R:OUT 9,0:NEXT R
```

### **9-2 Trying the noise generator**

This enables noise on channel A. Change the final value 8 in line 20 within the range 1-31 to alter the noise pitch:

```
10 FOR R=8 TO 10:OUT 8,R:OUT 9,0:NEXT R
20 OUT 8,6:OUT 9,8
30 OUT 8,7:OUT 9,55
40 OUT 8,8:OUT 9,15
```

### **9-3 Trying the envelope generator**

This returns channel A to a tone and applies a repeating triangular volume envelope. Writing register 13 restarts the envelope:

```
10 OUT 8,0:OUT 9,125:OUT 8,1:OUT 9,0
20 OUT 8,7:OUT 9,62
30 OUT 8,11:OUT 9,64:OUT 8,12:OUT 9,0
40 OUT 8,8:OUT 9,16
50 OUT 8,13:OUT 9,14
```

The complete register map, frequency formula, and an assembler example are in AY-3-8910 Programmable Sound Generator (#ay-3-8910-programmable-sound-generator).

## **10. FIRST STEPS WITH THE JOYSTICKS**

The two physical MEGA65 joystick sockets are connected to the parallel I/O ports of the emulated AY-3-8910. Port A is available in AY register 14 and port B in register 15. Both sockets support Up, Down, Left, Right, and Fire. The signals are active low: a pressed direction or button clears its corresponding bit.

To test the sockets, copy JOYTEST.NAS from the supplied NASCOM2 SD-card directory into /NASCOM2 on the MEGA65 SD card. Open the file browser with F11, load JOYTEST.NAS, return to NAS-SYS, and start it with:

```
E 1000
```

The live display has this form:

```
A:FF B:FF A[.....] B[.....]
```

U, D, L, R, and F replace the dots while the corresponding control is pressed. For example, Up on port A produces A:FE and A[U....]; Fire produces A:EF and A[....F]. Port B is shown independently in the second value and bracket. Press any keyboard key to leave the test.

R3/R3A and R6 use the same ten direct active-low FPGA inputs. On R6 the core also enables the shared cartridge/joystick interface and keeps all joystick drain outputs inactive. Standard Atari-style digital joysticks are supported; the four analogue paddle inputs are not currently exposed to Nascom software. The complete AY register and bit assignment is documented under AY-3-8910 Programmable Sound Generator (#ay-3-8910-programmable-sound-generator).

## **11. CP/M 2.2**

Even though the Nascom2 was a great and affordable machine for hobby users in the early 1980s, there was always an interest in looking beyond its original limits. It did not take long before people started developing solutions to get CP/M running on the Nascom.

Of course, this required adapted RAM/ROM management, a new ROM, and a few other small tricks, but some people were not afraid of the effort.

And neither was I. Although I could not find an original CP/M disk, I was able to create a working disk using ROM diagnostics.

USAGE: In the F13 configuration screen, CP/M can now be enabled below the FDC option. This automatically enables FDC and all RAM.

After pressing Reset, you should hopefully arrive at the CP/M boot screen. Now you only need to mount a CP/M 2.2 disk image, CPM22.DSK, on 0/1 and press Reset again. After a short boot time, you should end up in CP/M.

A first test is to enter DIR followed by Enter at the A> prompt, or alternatively STAT followed by Enter.

### **11-1 Commands & Tools on Disk CPM22.DSK**

- Built-In Commands:
- DIR - Produces a list of files matching the <filespec>, if no filespec is entered, the result is the same as ., i.e., all files on the currently logged disk. The command accepts similar wildcards as DOS.
- ERA - Deletes the file(s) specified by <filespec>. Wildcards are allowed.
- REN - REN <new> = <old> Renames the <old> filespec to the <new>. Wildcards are allowed.
- SAVE - Saves a copy of memory to file. <pages> is the number of 256 byte pages starting at address 100(h) to save and <file> is the filename to save the data to.
- TYPE - Copies the named text file to the console (screen)
- USER - The system has "user areas" numbered from 0 to 15. On start-up, the User area is set to "0". The DIR command displays files in the current user area and only programs in the current user area can be executed. The USER command is used to select another User area.
- COM-Files on Disk:
- FORMAT - Format a Disk in Drive A: or B:
- SYSGEN - Copy Bootfiles from A: to B:
- DISKCOPY - Copy a Disk from A: to B:
- TIME - Shows Date and Time from RTC
- AVCTEST - Displays a quick eight-colour AVC test pattern
- AVCDIAG - Tests all AVC RAM planes, scrolling and both resolutions
- AVCDEMO - Demonstrates the reusable pixel, line and rectangle routines
- KERMIT - universal file-transfer- and terminal-program for serial communication
- ASM - CP/M Assembler V2.0
- MAC - CP/M MAC Macro Assembler
- DDT - Dynamic Debugging Tool V2.2
- DUMP - Dumps memory on the screen
- ED - Standard CP/M Editor, don't use it. Take TE!
- LOAD - reads the file <filespec>, which is assumed to contain HEX format machine code, and produces a memory image file that can subsequently be executed
- PIP - Peripheral Interchange Program, used for, e.g., copying files. e.g. PIP A:=B:\*.COM Copies all .COM files on B: to A:
- POWER - small utility program that adds simple automation, macros, and startup control to the otherwise minimal CP/M command environment

- STAT - The STAT program can be used to display and change serial device, disk and file settings
- SUBMIT - allows CP/M commands to be batched for automatic processing
- TE - A very nice text editor optimized for the standard Nascom-screen 48x16
- XSUB - Extended Submit Facility loads below ccp, and feeds command lines to programs which read buffered input

### **11-2 Network Disk CPMNET.DSK**

CPMNET.DSK is a small bootable CP/M 2.2 disk intended for using the core's Ethernet interface directly from CP/M. It contains PING.COM, TELNET.COM and FTP.COM, plus the basic utilities PIP, STAT, DUMP and DDT. NETINFO.TXT provides a short on-disk command reference.

The CP/M network programs use the same ports \$B8/\$B9 and the same configured IP address, gateway and DNS server as the NASSYS programs. At the A> prompt, use PING host, TELNET host port, or FTP host port. FTP then asks only for the user name and password. Host names are resolved through the configured DNS server. The CP/M FTP client stores downloads as CP/M files and uploads CP/M files; local names must fit the 8.3 convention.

Because CP/M 2.2 records file sizes in 128-byte records, an upload includes the complete final record and any padding it contains.

## **12. FIRST STEPS WITH AVC**

The Lucas Logic Advanced Video Controller Model B was a popular graphics expansion for NASCOM computers. It adds three independent bitmap planes for red, green, and blue without replacing the standard NASCOM text display. A program can select either display at any time, so the normal monitor, ROM BASIC, NAS-DOS, and CP/M remain available.

The supported display modes are:

| Mode | Resolution               | Colours                  | Typical use               |
|------|--------------------------|--------------------------|---------------------------|
| 0    | Standard NASCOM video    | Normal text display      | Return from AVC graphics  |
| 1    | 392 x 256                | Eight colours            | General colour graphics   |
| 2    | 784 x 256                | Two colours              | High-resolution drawing   |
| 3    | 784 x 256 plus 392 x 256 | Black, blue, green, cyan | Mixed-resolution graphics |

The HDMI output remains 800 x 600 at 60 Hz. Double-density graphics are shown at their full horizontal resolution. The normal video-size setting can enlarge single-density AVC graphics, while the complete image remains centred on the screen.

The 96-column CP/M terminal and monochrome AVC double-density mode follow the configured green or amber phosphor colour. AVC modes using the blue output remain true-colour modes and are not recoloured.

### **12-1 Enabling the AVC**

1. Press F13 to open the configuration menu and press F13 again to reach RAM/ROM 2/3.
2. Use the cursor keys to select AVC.

3. Use Left or Right to change the setting to ON.
4. Press Enter to save and apply the configuration. The core restarts with the AVC available.

The AVC requires a CPU speed of at least 2 MHz. Enabling AVC automatically changes a 1 MHz configuration to 2 MHz. While AVC remains enabled, the CPU speed menu offers only 2 MHz and 4 MHz.

AVC can be enabled independently of FDD and CP/M. To disable it, repeat the same procedure and select OFF.

### **12-2 Trying the AVC under CP/M**

Enable both CP/M-MODUS and AVC, mount CPM22.DSK as drive A, and reset the machine. CP/M then uses the AVC automatically as a 96-column by 24-line text terminal.

The supplied disk contains three AVC programs:

- AVCTEST displays a quick eight-colour test pattern. Press any key to return to CP/M.
- AVCDemo draws coloured pixels, lines, and rectangles. Press any key to return.
- AVCDIAG tests the three 16 KiB video planes and then shows RGB output, hardware scrolling, the blinking MC6845 cursor, and both graphics resolutions. Press a key to advance through the visual tests.

AVCDIAG overwrites the contents of AVC video memory, but it does not alter the mounted disk or normal program memory.

### **12-3 Using G48 with ROM BASIC**

G48.NAS is the original Lucas Logic graphics extension for a 48 KiB NASCOM system. It adds drawing, colour, text, polygon, windowing, and graphics disk commands to the built-in ROM BASIC.

Before loading G48:

1. Enable AVC and leave CP/M-MODUS off.
  2. On configuration page RAM/ROM 2/3, select RAM for \$B000-\$B7FF, \$B800-\$BFFF, and \$C000-\$CFFF.
  3. Press Enter to apply the settings.
  4. Make sure G48.NAS is present in the /NASCOM2 directory of the SD card.
- Initialize ROM BASIC once with J. Accept the default memory size by pressing Enter, then use MONITOR to return to NAS-SYS. Load G48.NAS through the F11 file browser and start it as follows:

- Enter E CFFA for a system using NAS-SYS without NAS-DOS.
- For NAS-DOS, mount a disk and activate NAS-DOS with E D000 first. Then enter E CFFD.

G48 prints its initialization message and returns to the operating system.

Enter Z to warm-start ROM BASIC with the graphics commands installed.

All G48 graphics commands in ROM BASIC use the SET prefix. This small example selects the eight-colour mode, clears the screen, and draws three green lines:

```
10 SET MODE 1
20 SET BACKGND 0
```

```

30 SET COLOUR 2
40 SET PLOT 20,20
50 SET LINE 370,230
60 SET LINE 20,230
70 SET LINE 370,20
80 END

```

Run it with RUN. When the program ends, the keyboard is still active even though the graphics display remains selected. Enter the following command to return to the normal NASCOM screen:

```
SET MODE 0
```

Useful G48 commands include:

- SET MODE selects the display mode.
- SET BACKGND clears the plotting window with a selected colour.
- SET COLOUR selects the drawing colour.
- SET PLOT, SET LINE, SET RECT, SET TRI, and SET POLY draw graphics.
- SET LABEL draws scalable and rotatable text.
- SET WINDOW defines a clipping window.
- SET CHECK reads the colour of a pixel.
- SET GSAVE "NAME",0 saves a compressed picture to NAS-DOS drive 0.
- SET GLOAD "NAME",0 loads a previously saved picture.

GSAVE and GLOAD require FDD/NAS-DOS and a mounted disk. A G48 picture name may contain one to eight characters. GSAVE expects a new name and does not overwrite an existing file.

Assembly-language programs can control the card directly through ports \$B0-\$B2; their register definitions are listed in the Z80 I/O Ports section. The supplied CP/M test and demonstration programs provide working examples of initialization, screen clearing, pixels, lines, and rectangles.

### **13. NAS-DOS ROM PATCH**

The patch described here affects the 4 KB NAS-DOS/FDC ROM which is mapped at \$D000-\$DFFF when FDD support is enabled.

The unmodified ROM has the following entry in its public jump table:

```
D009: C3 90 D0    JP $D090
```

\$D009 is the documented entry point for formatting a disk. The original entry jumps directly to the internal formatter at \$D090. That formatter assumes that the FDC and drive 0 have already been initialized, for example by an earlier directory command. This assumption is normally true with a physical drive, but it is not guaranteed after mounting a .DSK image in the MEGA65 file browser. Consequently, the first format command could fail unless a command such as JD had been executed beforehand.

The generated ROM changes the jump at \$D009 and uses seven otherwise unused \$FF bytes at \$D3F0-\$D3F6:

```
D009: C3 F0 D3    JP $D3F0
```

D3F0: CD 76 D0      CALL \$D076

D3F3: C0            RET    NZ

D3F4: C3 90 D0      JP     \$D090

\$D076 is the original NAS-DOS FDC initialization routine and \$D090 remains the formatter entry. A successful initialization therefore continues through the original formatter. If initialization fails, the non-zero FDC status is returned to the caller.

The original formatter saves the result of the raw-track phase in AF, creates the 16 directory sectors afterwards, and finally returns the earlier result. A failed directory write branches to \$D143:

```
D13A: 20 07          JR    NZ,$D143
D13F: DF 6A          RST   $18 / function $6A
D141: F1            POP   AF
D142: C9            RET
D143: EF 2D 00      RST   $28 / text "-"
D146: DF 6B          RST   $18 / function $6B
D148: F1            POP   AF
D149: 3E FF          LD    A,$FF
D14B: B7            OR    A
D14C: C9            RET
```

With the emulated WD1793, the raw tracks and all directory sectors can be committed successfully although this saved value is still \$FF. NAS-DOS then reports Fmt.Error after a complete format. The generated ROM makes the normal return deterministic and redirects a real directory-write failure to the original compact \$FF return at \$D148:

```
D13A: 20 0C          JR    NZ,$D148
D141: F1            POP   AF
D142: AF            XOR   A
D143: C9            RET
```

This forces a zero result only after the complete 16-sector directory loop. An actual directory-write error still balances the saved stack value and returns \$FF through the unchanged code at \$D148-\$D14C. Only its preceding low-level "-" diagnostic is skipped; the caller still prints Fmt.Error. Compatibility details:

- The documented format entry point remains \$D009.
- All other public NAS-DOS entry points and their addresses remain unchanged.
- The formatter entry remains at \$D090; no public code or jump-table entry is relocated. Its internal return sequence at \$D13A-\$D143 is patched as described above.
- Byte-for-byte, only \$D00A-\$D00B, \$D13B, \$D142-\$D143, and the previously unused bytes at \$D3F0-\$D3F6 differ from the source ROM. The \$C3 jump opcode at \$D009 remains unchanged.

- The original source image  
 helper/NASCOM2-AllFiles/Nasdos.nas remains unchanged. The patch is applied reproducibly by helper/generatenasdosrom.ps1 while generating  
 Nascom2.srscs/sources1/new/nasdosromimagepkg.vhd.  
 The WD1793 WRITE TRACK implementation and the physical handling of NAS-DOS's 16 post-format directory initialization writes are FPGA controller behavior in Nascom2.vhd.  
 For hardware diagnosis, port \$E6 contains the retained format/backend status, \$E7 contains the track ID of the last completely committed WRITE TRACK, and \$E8 contains the number of post-format directory writes still expected. Ports \$E6-\$E8 are emulator extensions and are not part of the original Lucas controller.  
 For the post-format sequence, \$E6=\$40 means that the final committed track is waiting for the directory-clear sequence, \$41 means that sector 1 was accepted as its start, \$51-\$5F identify completed sectors 1-15, and \$60 means that all 16 directory initialization sectors were committed to the mounted image.

#### **14. REAL-TIME CLOCK SUPPORT**

The I2C bus already existed back in the 1980s, and one popular expansion for early computers was to add a real-time clock.

Back then, I built a solution using an MM58174 and successfully used it with my Nascom2. Fortunately, the MEGA65 already has a real-time clock connected via I2C, so I integrated it into the core.

On the SD card, there is a file named RTC.NAS. Load it with F11 and start it with E 1000 to display the current time and date.

The core supports both the onboard real-time clock and the optional external DS3231 module on the Grove connector on R3/R3A and R6 machines. Both I2C buses are checked independently. After repeated valid DS3231 readings, the external clock is selected automatically; if it is absent or stops responding, the core falls back to the onboard clock. The Grove clock therefore takes precedence when fitted, while machines without one continue to use their onboard RTC. No RTC address, jumper, or configuration-menu setting is required.

#### **15. PMOD RS232 UART COMMUNICATION**

The JTAG UART connection was nice, but it does not provide a way to communicate with old computers. Therefore, I added support for the PMOD RS232 module.

The module is available at a moderate price, for example in Germany from Trenz or Reichelt. It can also be built manually. The schematic is easy to find at Digilent, and it is not rocket science.

The MEGA65 has two PMOD connectors, each with 2 x 6 pins and 8 freely programmable signals. The remaining 4 pins are 3.3 V power and GND. This means there are four possible connection positions for a module with 6 connector pins, and all of them can be used, selectable via the menu.

##### **15-1 PMOD RS232 Pinout**

- Pin 1: CTS

- Pin 2: RTS
- Pin 3: TXD
- Pin 4: RXD
- Pin 5: GND
- Pin 6: VCC

## **16. ETHERNET AND NETWORK PROGRAMS**

Ethernet is available on MEGA65 R3/R3A and R6. It is optional and disabled by default. The core supports Ethernet II, ARP, static IPv4 or DHCP, ICMP echo requests and replies, DNS A-record lookups, and outbound TCP streams. IPv6, UDP applications, TLS, and active-mode FTP are not currently supported.

### **16-1 Network configuration**

Press F13 three times to open NETWORK 3/3. The page contains:

- NETWORK: enable or disable the network interface
- ADDRESS MODE: set static- or DHCP-mode
- IP ADDRESS: the static address of the Nascom; select DHCP as the address mode to obtain the IP settings automatically
- NETMASK: the local IPv4 subnet mask
- GATEWAY: the router used for destinations outside the local subnet
- DNS SERVER: the resolver used for host names
- LINK: the current physical Ethernet link state

Use Up/Down to select a row. On an address row, use Tab to select an octet and Left/Right to change it. Press Enter to apply and save the complete configuration. The values become active immediately; a reset or a new bitstream load is not required.

Use an unused address in the same subnet as the local network. For example, a Nascom connected to a 192.168.4.0/24 network could use:

```
IP ADDRESS  192.168.4.188
NETMASK     255.255.255.0
GATEWAY     192.168.4.1
DNS SERVER  192.168.4.1
```

The DNS server may be outside the local subnet as long as the gateway can route to it. The interface uses the fixed locally administered MAC address 02:4E:41:53:32:36.

After enabling the network, check that LINK UP is displayed. A computer on the same network should then be able to ping the configured Nascom address. This incoming ping test is a convenient first check before using the supplied network programs.

### **16-2 PING.NAS**

PING.NAS sends three ICMP echo requests from the Nascom.

1. Load PING.NAS with F11.
2. Start it from NASSYS with E 6000.
3. Enter a dotted IPv4 address or a host name.

Host names are resolved through the configured DNS server. The program prints the result of each of the three attempts and then returns to NASSYS.

### **16-3 TELNET.NAS**

TELNET.NAS is an interactive Telnet client with basic option negotiation and local echo support.

1. Load TELNET.NAS with F11.
2. Start it with E 3000.
3. Enter a server name or IPv4 address and a decimal TCP port.

Pressing Enter at a prompt accepts the displayed default. Press Esc during a session to close the connection and return to NASSYS. Return is transmitted as the Telnet CR/LF sequence. Local echo is used for simple TCP listeners and is disabled automatically when a Telnet server negotiates server-side echo.

### **16-4 FTP.NAS**

FTP.NAS is a passive-mode, binary FTP client.

1. Load FTP.NAS with F11.
2. Start it with E 5000.
3. Enter the server name or IPv4 address, control port, user name, and password.

Pressing Enter accepts the displayed default for an input field. Server names are resolved through DNS. After login, these commands are available at the FTP> prompt; the first letter is sufficient:

```
| Key | Function |
|-----|-----|
| D | Display a directory listing with size, date/time, and name |
| G | Download a remote file into Nascom memory |
| U | Upload a Nascom memory range as a remote file |
| C | Change the server directory |
| P | Display the current server directory |
| H | Display help |
| Q | Close the connection and return to NASSYS |
```

Directory output pauses after twelve entries. Downloads ask for a destination address and report the first free address after the received data. Upload addresses and lengths are entered in hexadecimal without a prefix. Do not download data over the FTP client itself at \$5000-\$5EFA.

FTP uses one TCP connection for commands and a second connection for each directory listing, download, or upload. A server firewall must therefore allow the FTP control port, normally TCP 21, and the server's configured passive port range. If login works and a 227 response is shown but a transfer fails, check the passive port range and its firewall rules first.

### **16-5 CONTROL.NAS**

CONTROL.NAS is intended for remote-control and GUI applications. It allows a computer to read and write Nascom memory, access I/O ports, and start Z80 code through a simple TCP protocol.

Load it with F11 and start it with E 4000. Enter the IPv4 address of the

computer running the control application. The computer must listen on TCP port 2323 before the agent is started. The Nascom initiates the connection. Use the QUIT operation in the control application for an orderly disconnect and return to NASSYS.

#### **16-6 Notes and troubleshooting**

- LINK DOWN: check the Ethernet cable and switch port.
- DNS names fail but numeric addresses work: verify the DNS server and gateway.
- Local addresses fail after routed addresses were used: re-enter the intended server address and port; pressing Enter accepts the value currently shown.
- FTP login works but transfers fail: allow the passive data-port range on the server firewall.
- A TCP connection that is still closing can take a short time before a new connection starts. The clients close their sockets during startup and exit to make repeated runs reliable.
- The FPGA TCP receive window is intentionally small. Transfers are reliable but are not intended to match modern computer network speeds.

### **17. CONFIGURATION FILE N2CONF.CFG**

The file is stored as /NASCOM2/N2CONF.CFG on the FAT32 SD card. It is loaded automatically after SD-card initialization and saved when configuration changes are applied with ENTER. A newly created file receives FAT attributes \$22 (hidden + archive). When an existing legacy file is upgraded, its directory size is changed to 32 bytes and the current V4 data replaces its first sector.

#### **17-1 Current Format**

The current writer stores N2CONF.CFG in a 32-byte format.

Bytes 0..3 are the fixed signature:

4E 32 43 46

This is ASCII N2CF.

When loading, the first 4 bytes must match exactly:

readfirstword\_reg = x"4E324346"

In addition, the loader checks:

- file size >= 32 bytes
- byte 4 = format version 04
- byte 5 = fixed marker 20
- byte 30 = XOR checksum

#### **17-2 Byte Layout**

| Byte | Name      | Meaning |
|------|-----------|---------|
| 0    | signature | 4E = N  |
| 1    | signature | 32 = 2  |
| 2    | signature | 43 = C  |
| 3    | signature | 46 = F  |
| 4    | version   | 04      |

```

| 5 | marker | 20 |
| 6 | cfg_byte4 | speed, phosphor, scanlines, video zoom |
| 7 | cfg_byte5 | slot ROM mask + FDD/NAS-DOS + CP/M mode |
| 8 | cfg_byte6 | RS232 selection + RTS/CTS |
| 9 | cfg_byte8 | RS232 speed |
| 10 | network flags| bit 0 = Ethernet enabled, bit 1 = DHCP enabled |
| 11-14| IPv4 address | network byte order |
| 15-18| netmask | network byte order |
| 19-22| gateway | network byte order |
| 23-26| DNS server | network byte order |
| 27-29| reserved | 00 |
| 30 | checksum | XOR checksum |
| 31 | terminator | 0A |

```

### **17-3 Bit Layout**

#### **17-4 Byte 6 = cfg\_byte4**

- Bits 7 downto 6 = 0 (for future use)
- Bits 5 downto 4 = video zoom (00 = original, 01 = 150%, 10 = 200%)
- Bit 3 = cfgsavescanlines
- Bit 2 = cfgsavephosphor\_amber
- Bits 1 downto 0 = cfgsavespeed

In practice:

00 ZZ S P SS

- ZZ = video zoom (00 original, 01 150%, 10 200%)
- S = scanlines
- P = phosphor (0 = green, 1 = amber)
- SS = CPU speed

#### **17-5 Byte 7 = cfg\_byte5**

- Bits 3 downto 0 = cfgsaveslot\_rom
- Bit 4 = cfgsavefdd
- Bit 5 = cfgsavecpm
- Bit 6 = cfgsaveavc
- Bit 7 = cfgsavebls

In practice:

B A C F RRRR

- F = FDD/NAS-DOS enable
- C = CP/M mode; forces FDD on and all configurable ROM slots to RAM
- A = AVC enable
- B = BLS Pascal V1.3 mode
- RRRR = ROM/RAM slot mask

The ROM mask bits select:

```

| Bit | Address range | 0 | 1 |
|-----|-----|-----|-----|
| 0 | $B000-$B7FF | RAM | TOOLKIT ROM |

```

```
| 1 | $B800-$BFFF | RAM | NASPEN ROM |
| 2 | $C000-$CFFF | RAM | SUPERDEBUG/NAS-DIS ROM |
| 3 | $D000-$DFFF | RAM | ZEAP Assembler ROM |
```

When FDD bit 4 is 1, NAS-DOS is mapped at \$D000-\$DFFF regardless of ROM-mask bit 3. The saved value of bit 3 is retained, but it only becomes effective again after FDD is disabled.

When CP/M bit 5 is 1, the runtime configuration forces FDD on and clears all four live ROM-mask bits. The configured mask is nevertheless retained in the file and restored when CP/M mode is disabled again. In CP/M mode \$D000-\$DFFF is writable RAM rather than the NAS-DOS ROM. Low and high shadow RAM plus the 2 KiB boot EPROM implement the preliminary CP/M boot memory view; the custom second-stage loader, BIOS, and system disk are still separate work. When BLS Pascal bit 7 is 1, the memory map is \$A000-\$CFFF = BLS Pascal ROM and \$E000-\$FFFF = RAM. At \$D000-\$DFFF, an enabled FDD keeps NAS-DOS visible; without FDD this range is RAM. The normal expansion-ROM mask remains saved and is restored when BLS Pascal is disabled. BLS Pascal and CP/M mode are mutually exclusive.

#### **17-6 Byte 8 = cfg byte6**

- Bits 2 downto 0 = cfigsavers232
- Bits 5 downto 3 = 0
- Bit 6 = cfigsavers232\_flow
- Bit 7 = 0

In practice:

0 F 000 RRR

- RRR = RS232 routing selection
- F = RTS/CTS flow control enable

#### **17-7 Byte 9 = cfg byte8**

- Bits 3 downto 0 = cfigsavers232\_speed
- Bits 7 downto 4 = 0

In practice:

0000 BBBB

- BBBB = RS232 baud-rate selector

#### **17-8 Encoded Values**

CPU speed in byte 6, bits 1:0:

| Value | Effective speed                                       |
|-------|-------------------------------------------------------|
| 00    | 1 MHz                                                 |
| 01    | 2 MHz                                                 |
| 10    | approximately 4.166 MHz, displayed as 4 MHz           |
| 11    | approximately 4.166 MHz; accepted as a fallback value |

RS232 routing in byte 8, bits 2:0:

| Value | Route |
|-------|-------|
| 000   |       |

|         |                                                   |
|---------|---------------------------------------------------|
| 000     | JTAG UART                                         |
| 001     | PMOD 1 High                                       |
| 010     | PMOD 1 Low                                        |
| 011     | PMOD 2 High                                       |
| 100     | PMOD 2 Low                                        |
| 101-111 | reserved; not generated by the configuration menu |

For the JTAG route, the UART always runs at 115200 baud and RTS/CTS is not used. The stored baud and flow-control bits are relevant to the PMOD routes.

Baud-rate selector in byte 9, bits 3:0:

|           |                                    |
|-----------|------------------------------------|
| Value     | Baud                               |
| -----     | -----                              |
| 0000      | 300                                |
| 0001      | 1200                               |
| 0010      | 2400                               |
| 0011      | 4800                               |
| 0100      | 9600                               |
| 0101      | 14400                              |
| 0110      | 19200                              |
| 0111      | 38400                              |
| 1000      | 57600                              |
| 1001      | 115200                             |
| 1010-1111 | accepted as 115200 fallback values |

#### **17-9 Checksum**

Byte 30 contains the XOR of bytes 4 through 26 and the fixed value A5.

Bytes 27 through 29 are currently zero and do not contribute a non-zero value.

version XOR marker XOR configuration payload bytes 6..26 XOR x"A5"

That is:

Byte30 = Byte4 XOR Byte5 XOR ... XOR Byte26 XOR A5

using the file-byte numbering.

Byte 31 is written as 0A, but the loader does not currently use it for validation.

#### **17-10 Values Read Back**

When loading the current V4 format, the original settings are decoded as before. In addition, byte 10 selects Ethernet (bit 0) and DHCP mode (bit 1), while bytes 11 through 26 retain the static IPv4 address, netmask, gateway, and DNS server. DHCP lease values are obtained at runtime and are not saved.

#### **17-11 From Byte 6**

- speed <= bits 1:0
- phosphor <= bit 2
- scanlines <= bit 3
- video\_zoom <= bits 5:4

#### **17-12 From Byte 7**

- slot\_rom <= bits 3:0

- fdd <= bit 4
- cpm <= bit 5
- avc <= bit 6

#### **17-13 From Byte 8**

- rs232 <= bits 2:0
- rs232\_flow <= bit 6

#### **17-14 From Byte 9**

- rs232\_speed <= bits 3:0

#### **17-15 Backward Compatibility**

The loader still accepts older config formats:

- Legacy 8-byte format:
- bytes 0..3 = N2CF
- byte 4 = speed and phosphor
- byte 5 = ROM-slot mask
- byte 6 = byte4 XOR byte5 XOR A5
- RS232 defaults to JTAG/115200; scanlines, zoom, RTS/CTS, and FDD default off
- Legacy 9-byte RS232 format:
- bytes 0..3 = N2CF
- byte 4 = speed and phosphor
- byte 5 = ROM-slot mask
- byte 6 = RS232 route in bits 2:0 and legacy baud selector in bits 5:3
- byte 7 = byte4 XOR byte5 XOR byte6 XOR A5
- scanlines, zoom, RTS/CTS, and FDD default off
- Version 2 of the 16-byte format:
- byte 4 = 02, byte 5 = 10
- byte 6 also contains scanlines in bit 3
- byte 7 contains only the four ROM-slot bits
- byte 8 contains RS232 route in bits 2:0 and legacy baud in bits 5:3
- byte 14 = byte4 XOR byte5 XOR byte6 XOR byte7 XOR byte8 XOR A5
- zoom, RTS/CTS, and FDD default off
- Version 3 of the 16-byte format:
- byte 4 = 03, byte 5 = 10
- bytes 6 through 9 contain the current CPU/video/ROM/FDD/CP/M/AVC/RS232 settings
- byte 14 contains the V3 XOR checksum
- Ethernet defaults to off and the built-in static IPv4 values are used

The legacy three-bit baud selector used by the 9-byte and V2 formats is converted as follows:

| Legacy value | Loaded baud |
|--------------|-------------|
| ----- -----  |             |
| 000          | 115200      |
| 001          | 300         |
| 010          | 2400        |
| 011          | 9600        |

| 100-111 | 57600 |

However, new files are always written in the 32-byte V4 format described above. If no valid configuration file is loaded, the built-in defaults are 1 MHz, green without scanlines, original video size, JTAG UART at 115200 baud, all four optional ROM slots enabled, FDD/NAS-DOS disabled, CP/M disabled, and AVC disabled, Ethernet disabled, and DHCP disabled. The initial static network values are 192.168.1.65, netmask 255.255.255.0, gateway 192.168.1.1, and DNS 192.168.1.1.

## **18. Z80 I/O PORTS**

### **18-1 Port Changes from the Original CP/M Design**

| Port  | New assignment                   | Note                        |
|-------|----------------------------------|-----------------------------|
| ----- | -----                            | -----                       |
| \$B0  | AVC MC6845 address register      | Decoded only when AVC ON    |
| \$B1  | AVC MC6845 data register         | Decoded only when AVC ON    |
| \$B2  | AVC control register             | Decoded only when AVC ON    |
| \$B3  | AVC 96-column terminal character | Private FPGA extension      |
| \$B4  | AVC terminal control/status      | Private FPGA extension      |
| \$BF  | Internal CP/M bank switching     | Effective only in CP/M mode |

The historical CP/M boot ROM also used port \$B2 for memory bank switching.

That port now belongs exclusively to the AVC. The generated boot ROM, loader, and BIOS are therefore patched to use the private core port \$BF. CP/M software must use \$BF, rather than \$B2, for bank switching.

### **18-2 NASCOM Basic Functions**

| Port  | Access | Function                                 |
|-------|--------|------------------------------------------|
| ----- | -----  | -----                                    |
| \$00  | R/W    | NASCOM keyboard matrix and control latch |
| \$01  | R/W    | Serial data: read RX / write TX          |
| \$02  | R      | Serial status                            |
| \$BF  | W      | Internal CP/M memory/overlay switching   |

- When writing to \$00:

- Bit 0: advance the keyboard row

- Bit 1: reset the keyboard counter

- Bit 3: request NASCOM single-step/NMI

- Bit 4: TAPE LED

- The remaining bits currently have no functional effect.

- At \$02:

- Bit 7: RX character available

- Bit 6: transmitter ready

- At \$BF in CP/M mode:

- Bit 7: expose RAM at \$0000-\$0BFF

- Bit 6: disable the boot/video overlay at \$F000-\$FFFF

- \$80: low RAM active, upper boot/video overlay visible

- \$C0: full 64 KiB RAM view
- Writes to \$BF have no effect outside CP/M mode.

### **18-3 AY-3-8910 Programmable Sound Generator**

The core contains an AY-3-8910-compatible programmable sound generator. It runs from a fixed 2 MHz clock, independently of the selected Z80 speed. Its three channels are mixed to mono and transmitted as 48 kHz, signed 16-bit PCM on both HDMI audio channels. The same mono mix is also sent to the left and right channels of the MEGA65 3.5 mm audio jack. VGA output does not carry audio. The interface follows the direct NAS-BUS design described by Andy Tipler in INMC 80 News issue 5. It does not use the NASCOM PIO:

|Port|Access|Function|

|----|-----|-----|

|\$08|W|Select AY register 0-15; only bits 3-0 are used|

|\$09|R/W|Read or write the selected AY register|

For example, this Z80 sequence enables tone channel A at maximum volume:

```
LD A,0
OUT (08H),A      ; tone A fine period
LD A,125
OUT (09H),A
LD A,1
OUT (08H),A      ; tone A coarse period
XOR A
OUT (09H),A
LD A,7
OUT (08H),A      ; enable tone A, disable other tone/noise inputs
LD A,0FEH
OUT (09H),A
LD A,8
OUT (08H),A      ; channel A fixed volume
LD A,15
OUT (09H),A
```

The tone frequency is  $2,000,000 / (16 \times \text{period})$  Hz, where period is the 12-bit value in each channel's fine and coarse registers. A zero period is treated as one.

|Register|Function|

|-----|-----|

|0, 1|Channel A tone period, fine 8 bits and coarse 4 bits|

|2, 3|Channel B tone period|

|4, 5|Channel C tone period|

|6|Noise period, 5 bits|

|7|Tone/noise mixer disable bits and I/O direction bits|

|8-10|Channel A-C amplitude; bit 4 selects the envelope|

|11, 12|Envelope period, 16 bits|

|13|Envelope shape; every write restarts the envelope|

|14, 15|Parallel I/O port A/B; MEGA65 joystick A/B in input mode|

On the MEGA65, parallel ports A and B read the two physical joystick ports.

On R6 the joystick sockets are connected directly to the FPGA using the same ten input pins as R3/R3A. The shared cartridge/joystick interface is enabled, and its ten active-low drain outputs are held inactive because this core uses the sockets strictly as inputs. The R3 board wrapper exposes the same ten active-low inputs without the R6 interface-control signals.

The five low bits use the usual active-low digital joystick layout:

|Bit|Input|

|---|-----|

|0|Up|

|1|Down|

|2|Left|

|3|Right|

|4|Fire|

|7-5|Unused, read as 1|

Register 7 bit 6 selects port A and bit 7 selects port B (0 = input, 1 = output latch). Both bits are 0 after reset, so software can select register 14 or 15 via port \$08 and immediately read the joystick at \$09.

For example, 0FEH means Up is pressed and 0EFH means Fire is pressed.

Writes to registers 14 and 15 remain available as output latches when the corresponding direction bit selects output mode.

The four analogue paddle inputs are not part of this digital AY mapping. They need pulse/discharge timing and separate result registers and are therefore not enabled by the joystick interface.

#### **18-4 Lucas Logic Advanced Video Controller (AVC Model B)**

The AVC is disabled by default and can be enabled independently of CP/M on the second F13 configuration page. Changing the setting is persisted and takes effect with the same controlled reset used for memory-map changes.

|Port|Access|Function|

|----|-----|-----|

|\$B0|W|Select an MC6845 register|

|\$B1|R/W|Read/write the selected MC6845 register|

|\$B2|R/W|AVC control register|

|\$B3|R/W|FPGA terminal character data|

|\$B4|R/W|FPGA terminal control/status|

- Bits 0-2 of \$B2: map the red, green, and blue 16 KiB planes into \$8000-\$BFFF. Writes can target multiple planes simultaneously; reads use the first selected plane.

- Bit 3: double-density mode, 784 × 256 instead of 392 × 256 pixels.

- Bits 4-6: enable red, green, and blue video output.

- Bit 7: display normal NASCOM video instead of AVC output.

- Reset safely initializes \$B2 to \$80: no plane is mapped and normal NASCOM video is visible.
  - With AVC OFF, \$B0-\$B2 are completely disabled: reads return \$00, writes are ignored, the \$8000-\$BFFF plane window is not mapped, and normal NASCOM memory remains visible there.
  - AVC and CP/M can operate together when AVC ON is selected. Mapping a color plane through \$B2 temporarily occupies \$8000-\$BFFF in the CPU address space, just as on the original hardware.
  - In CP/M mode, an enabled AVC is automatically used as a 96 x 24 terminal. The renderer joins the red and green planes in 784 x 384 double-density mode and draws the existing 8 x 16 core font into a 768 x 384 text area. The FPGA terminal extends each internal AVC plane to 32 KiB; historical AVC software retains its original 16 KiB CPU window and 256-line raster. The core uses an 800 x 600p60 HDMI mode so all 784 AVC pixels are displayed one-for-one. The normal video-zoom setting is deliberately ignored for this terminal mode so no columns are cropped. Carriage return, line feed, backspace and form feed are handled in hardware. Scrolling advances the MC6845 start address through a circular framebuffer.
  - \$B3 is a private character FIFO with one pending character. Software must wait until \$B4 bit 0 is one before writing. \$B4 bit 1 reports that the terminal is enabled. Writing bit 0 as one enables and clears the terminal; writing zero disables it and returns to normal NASCOM video. The MC6845 hardware cursor output is also implemented. On the physical Model B the CURSOR signal was available only at a test point and had to be connected to a chosen video output, normally through an XOR gate. To leave historical software unchanged, the MEGA65 implementation uses the otherwise unused bit 7 of MC6845 register 10 as this virtual link:
    - Register 10 bit 7: connect the cursor XOR output (1) or leave it disconnected (0).
    - Register 10 bits 6-5: steady, hidden, blink every 16 fields, or blink every 32 fields.
    - Register 10 bits 4-0 and register 11 bits 4-0: first and last cursor raster line.
    - Registers 14 and 15: 14-bit cursor address.
- The cursor is disabled in the private 96 x 24 CP/M terminal renderer, which maintains its own character cursor behaviour. Light-pen registers 16 and 17 remain passive readback registers; no light pen is emulated.

#### **18-5 Original G48 ROM BASIC graphics software**

The SD-card contents include the Lucas Logic G48 graphics extension for ROM BASIC. It occupies \$AEF6-\$CFFF and provides both NAS-SYS and NAS-DOS startup entries as well as the documented assembler interface. See [First steps with AVC \(#first-steps-with-avc\)](#) for memory configuration, startup, example BASIC code, and the available graphics commands.

#### **18-6 Real-Time Clock**

| Port | Access | Function     |
|------|--------|--------------|
| ---- | -----  | -----        |
| \$10 | R      | Seconds, BCD |

```

|$11|R|Minutes, BCD|
|$12|R|Hours, BCD|
|$13|R|Day of week|
|$14|R|Day, BCD|
|$15|R|Month, BCD|
|$16|R|Year, BCD|
|$17|R|RTC status|
|$18|R|I2C debug: busy counter|
|$19|R|I2C debug: status|
|$1A|R|I2C debug: last byte|
|$1B|R|I2C debug: last register address|
- Status at $17:
- Bit 7: RTC data valid
- Bit 6: access busy
- Bit 5: error
- Bit 4: update toggle
- Bits 3-0: error code

```

#### **18-7 WD1793/Floppy Controller**

```

|Port|Access|Function|
|----|-----|-----|
|$E0|R/W|Read WD1793 status / write command|
|$E1|R/W|Track register|
|$E2|R/W|Sector register|
|$E3|R/W|FDC data register|
|$E4|R/W|Drive control latch|
|$E5|R|DRQ, INTRQ, and drive-ready status|
|$E6|R|Format diagnostic code|
|$E7|R|Last formatted track|
|$E8|R|Remaining directory writes|
- Control latch at $E4:
- Bit 0: drive A
- Bit 1: drive B
- Bit 4: side 1
- Bit 5: motor on
- Status at $E5:
- Bit 7: DRQ - data byte ready/requested
- Bit 1: NOT READY
- Bit 0: INTRQ - command completed
- Main commands supported at $E0:
- Restore
- Seek
- Step / Step In / Step Out
- Read Sector / Read Multiple
- Write Sector / Write Multiple

```

- Read Address
- Write Track
- Force Interrupt

### **18-8 CP/M/FDC Diagnostic Ports**

These write ports are used exclusively by the CP/M BIOS to latch a read error for the F7 diagnostics page:

```
|Port|Access|Diagnostic value|
|----|-----|-----|
|$E7|W|Bytes still expected|
|$E8|W|CP/M track|
|$E9|W|CP/M record|
|$EA|W|WD1793 status at the time of failure|
```

- \$E7 and \$E8 therefore have different meanings for reads and writes.

The table covers 32 distinct Z80 port addresses. With AVC OFF, the five AVC addresses \$B0-\$B4 are inactive. Ports not listed, or disabled by the current configuration, return \$00 when read and ignore writes.

## **19. SOURCES AND ACKNOWLEDGEMENTS**

Thanks to the contributors of the Tyto2 and mega65-core projects.

Their work helped me learn how to use HDMI, VGA, keyboard input, and SD card support.  
<<https://github.com/amb5l/tyto2>> <<https://github.com/amb5l/mega65-core>>

Thanks to Daniel Wallner <[jesus@opencores.org](mailto:jesus@opencores.org)> for his T80s Core

Many stuff and inspiration on <<http://www.nascomhomepage.com/>>

The Lucas Logic AVC Model B implementation follows the original

hardware manual (<https://nascom.wordpress.com/wp-content/uploads/2018/06/avchman.pdf>)

and colour graphics software manual

(<https://nascom.wordpress.com/wp-content/uploads/2018/06/avcsman.pdf>).

Searchable OCR transcripts of both scanned manuals are kept under

helper/avc-docs; consult the scans where OCR text is ambiguous.

Finally, a huge thank you to the MEGA65 team!

It is a fantastic machine with so many possibilities on so many levels. I can only recommend it to anyone interested in vintage computers.

It is not perfect, but it is pretty close! ;)

## **20. LICENSING**

This file is part of the MEGA65-Nascom2-Emulator.

The MEGA65-Nascom2-Emulator is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License.

The MEGA65-Nascom2-Emulator is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with the MEGA65-Nascom2-Emulator.

If not, see <http://www.gnu.org/licenses/>.

## **21. AUTHOR**

April - September 2026

Thomas Linke, Ruhrpott, Germany

Email: [<mega65@nascom2.de>](mailto:mega65@nascom2.de)

Website: [<https://nascom2.de>](https://nascom2.de) and [<https://mega65.nascom2.de>](https://mega65.nascom2.de)

## ADDENDUM 01

### Nascom network I/O ABI

The R3/R3A and R6 network interface is shared by NASSYS and CP/M and uses two Z80 I/O ports. The interface is polling-based; neither operating environment needs an interrupt handler.

| Port | OUT                       | IN                        |
|------|---------------------------|---------------------------|
| B8h  | command                   | status                    |
| B9h  | parameter or payload byte | response or received byte |

Status bits read from B8h:

| Bit | Meaning                          |
|-----|----------------------------------|
| 7   | interface present                |
| 6   | Ethernet link up                 |
| 5   | network enabled in configuration |
| 4   | response byte available at B9h   |
| 3   | command/parameter writer ready   |
| 2   | requested DNS result ready       |
| 1   | requested DNS operation busy     |
| 0   | requested DNS operation failed   |

ABI version 4 commands:

| Command             | Response bytes                                                                                                        |
|---------------------|-----------------------------------------------------------------------------------------------------------------------|
| 00h RESET           | none; clears the CPU-side transaction                                                                                 |
| 01h GET_INFO        | 4Eh 32h, ABI version, capability bits                                                                                 |
| 02h GET_CONFIG      | flags, IPv4, netmask, gateway, DNS (network byte order)                                                               |
| 03h GETDNSRESULT    | result code followed by four result/status bytes                                                                      |
| 04h START_DNS       | parameters: DNS wire-format QNAME (3..64 bytes including final zero);<br>response: result                             |
| 05h START_PING      | parameters: target IPv4 (4, network byte order); response: result                                                     |
| 06h GETPINGRESULT   | one byte: 01h busy, 00h reply, E1h ARP error, E2h echo timeout                                                        |
| 10h SELECT_SOCKET   | parameter: socket 00h..03h; response: result, socket                                                                  |
| 11h CONFIG_SOCKET   | parameters: IPv4 (4), TCP port (2); response: result                                                                  |
| 12h CONNECT         | connect selected configured socket; response: result                                                                  |
| 13h GETSOCKETSTATUS | socket, state, IPv4 (4), TCP port (2)                                                                                 |
| 14h SEND            | parameters: 1..128 stream bytes; response: result, accepted length                                                    |
| 15h GETRXCOUNT      | response: result, pending received byte count                                                                         |
| 16h RECEIVE         | response: result, length, then the complete received chunk                                                            |
| 17h CLOSE           | idempotent close of the selected socket; sends FIN/ACK when established<br>and also succeeds after a peer close/reset |
| 7Fh LOOPBACK        | returns bytes previously written to B9h (maximum 128)                                                                 |

Capability bit 0 is ICMP, bit 1 is DNS, bit 2 is TCP connect and bit 3 is TCP stream I/O. Four CPU-side sockets are available and keep independent TCP state, outstanding sends, and 128-byte receive buffers. The Ethernet transmitter is shared and serializes ready segments. Each SEND/RECEIVE chunk can contain up to 128 bytes. The advertised receive window is one chunk; while that chunk is waiting for RECEIVE the core advertises a zero window, then sends a window update after the Z80 consumes it. This permits simultaneous FTP command and PASV data connections without overwriting unread data.

Each new connection uses a fresh ephemeral local port and initial sequence number, so reconnecting cannot collide with a server-side TCP control block left from an earlier

session. Before sending its SYN, each socket resolves its own Ethernet next hop: same-subnet destinations are ARPed directly, while routed destinations use the configured gateway. Socket states returned by GETSOCKETSTATUS are 00h closed/unconfigured, 01h configured/inactive, 02h connecting, 03h established and 04h closing. E1h means that no usable ARP/DNS route was available and E2h is a handshake timeout. Result E1h from a command means invalid parameter length/value; E2h means the socket, link or network configuration is not ready. A CONNECT request received while the previous singleton connection is still closing is queued and starts only after that close has completed. SEND result E3h means the socket is closed, a previous send is awaiting acknowledgement, or the supplied payload is empty. CLOSE is idempotent, including after a peer reset. RECEIVE result 01h means no chunk is ready.

START\_DNS permits runtime A-record lookups. Labels are length-prefixed in the usual DNS wire format; for example ftp.local is sent as 03h "ftp" 05h "local" 00h. DNS result code 00h means success. 01h is starting, 02h is gateway ARP, 03h is waiting for DNS, and E1h..E3h are the displayed DNS errors. The resolver walks the complete answer section and therefore accepts CNAME chains followed by an IPv4 A record. E3h means that the reply was malformed, reported a DNS error, or contained no usable A record.

### **NASSYS Ping client**

PING.NAS is loaded at 6000h and started with E 6000. It asks for one dotted IPv4 address or host name, resolves host names through START\_DNS, and sends exactly three ICMP echo requests. Each attempt prints REPLY, TIMEOUT or an error and the program then returns to NASSYS.

### **NASSYS Telnet client**

TELNET.NAS is a small interactive ABI V4 client loaded at 3000h and started with E 3000. It prompts for a host name or dotted-decimal IPv4 address and a decimal TCP port; host names are resolved through START\_DNS. Empty input selects the defaults 192.168.4.123 and 23.

It uses socket 3. Escape performs an orderly CLOSE and returns to NASSYS. The client polls NASSYS input without blocking network reception, translates Return to the NVT CR/LF pair, accepts server ECHO and SUPPRESS-GO-AHEAD, accepts client SUPPRESS-GO-AHEAD, rejects unsupported Telnet options, and skips Telnet subnegotiation blocks. Local echo is enabled for simple listeners and is automatically disabled while the server negotiates the Telnet ECHO option.

### **NASSYS FTP client**

FTP.NAS is loaded at 5000h and started with E 5000. It prompts for the server host name or IPv4 address, command port, user name, and password; empty input uses 192.168.4.123, port 21, anonymous, and nascom@. Socket 0 remains the FTP command connection and socket 1 is opened for each passive-mode transfer. The client selects binary TYPE I and supports directory listing with compact size/date/name output and paging after twelve entries, download to memory, upload from memory, CWD, PWD, and orderly QUIT. See Readme\_FTP.md for the command keys and the CONTROL-assisted loading workflow.

## ADDENDUM 02

### NASCOM control agent

helper/CONTROL.NAS is loaded at 4000h and started with E 4000. It prompts for the host controller's IPv4 address, using 192.168.4.123 for empty input, and connects socket 2 to fixed TCP port 2323. The resident image, buffers, and private stack occupy 4000h through 46DCh.

Start the host first. By default it binds only to the development PC's LAN address 192.168.4.123:

```
python helper\nascom_control.py
```

For the first hardware check, python helper\nascom\_control.py --smoke-test automatically verifies ping, ROM reading, reversible RAM write/read at 4700h, and input port 02h, then closes the agent and returns it to NASSYS.

Then load and execute CONTROL.NAS, enter the listener's IPv4 address (or press Return for 192.168.4.123), and let it connect. The host accepts these interactive commands:

```
ping
read ADDRESS LENGTH
write ADDRESS HEXBYTES
load FILE ADDRESS
save FILE ADDRESS LENGTH
in PORT
out PORT VALUE
exec ADDRESS
quit
```

Numbers may use 0x1234, \$1234, 1234h, or decimal notation. Transfers are split automatically into blocks of at most 120 bytes. Writes overlapping the resident agent are rejected by the host unless it is started with --force-agent-write. exec behaves like an indirect call: control returns to the agent when the injected routine executes RET. quit closes the TCP connection and returns the NASCOM to NASSYS.

#### **Binary framing**

A request is:

```
4E 32 CMD SEQ ARG_H ARG_L LEN [PAYLOAD] XOR
```

A response is:

```
32 4E (CMD|80) SEQ STATUS LEN [PAYLOAD] XOR
```

The XOR covers CMD through the byte before it; including the appended XOR in the calculation therefore produces zero. Commands 11h (memory write) and 21h (port output) carry request payload. For 10h (memory read), LEN is the requested response length. Status values are 00h success, 01h unknown command, 02h invalid length, and 03h checksum error.

## ADDENDUM 03

### MEGA65 R6 NASCOM Ping

PING.NAS is a simple ICMP test for NASSYS. The program is located at 6000h and is started with:

E 6000

At the TARGET IP/HOST: prompt, enter either an IPv4 address or a host name. Host names are resolved as A records through the DNS server configured on the network configuration page.

The program sends exactly three ICMP echo requests. Each attempt produces one of the following messages:

- REPLY - a matching echo reply was received;
- TIMEOUT: E2 - no matching reply was received before the timeout;
- ERROR: E1 - the target or next hop could not be reached through ARP.

The program then displays DONE and returns to NASSYS. There are no additional options and no continuous mode.

The PING.bin binary occupies 6000h..6366h (871 bytes). Direct targets on the local subnet are resolved through ARP; remote targets use the configured gateway's MAC address.

## ADDENDUM 04

### MEGA65 R6 NASCOM Telnet

The Telnet client TELNET.NAS is loaded at 3000h and started with the following NASSYS command:

E 3000

At the SERVER/IP prompt, enter either an IPv4 address or a host name. Host names are resolved as A records through the DNS server configured on the network configuration page. Empty input uses the defaults 192.168.4.123 and port 23 on every new program start.

After the connection has been established, the client operates as before. Local echo is enabled for simple listeners and is disabled when the server negotiates Telnet ECHO. Escape closes the connection and returns to NASSYS.

The binary image occupies 3000h..36EFh (1776 bytes). DNS support requires the current bitstream with the ABI V4 commands STARTDNS and GETDNS\_RESULT.

## ADDENDUM 05

### MEGA65 R6 NASCOM FTP Client

The FTP client is provided as FTP.NAS and as the raw binary image FTP.bin. It requires at least core V1.7 with support for concurrent TCP sockets.

#### Loading and starting

To load the NAS file directly:

1. Load FTP.NAS with F11.
2. Start it with E 5000.

The Control Agent can load the smaller binary image in one operation, verify it byte for byte, and then terminate the agent:

```
python -u nascom_control.py --bind 192.168.4.123 `
--load-file FTP.bin --load-address 5000h`
```

The same operations are also available interactively:

```
load FTP.bin 5000
read 5000 10
quit
```

The program occupies 5000h through 5EFAh, inclusive (3835 bytes). After terminating the Control Agent, this memory range can be saved as FTP.NAS with the usual NASSYS facilities and subsequently started with E 5000.

#### Login

At startup, the program asks for the server name/IP, TCP port, user name, and password. A host name is resolved as an A record through the DNS server entered on the network configuration page. Empty input accepts the displayed default:

```
SERVER/IP [192.168.4.123]:
PORT [21]:
USER [anonymous]:
PASSWORD [nascom@]:
```

After login, the client selects binary transfer mode with TYPE I. All FTP responses, including multiline responses, are displayed.

#### Commands

At the FTP> prompt, the first letter is sufficient:

| Key | Function                                                              |
|-----|-----------------------------------------------------------------------|
| D   | Display a compact LIST directory containing size, date/time, and name |
| G   | Download a file into NASCOM memory with RETR                          |
| U   | Upload a memory range as a file with STOR                             |
| C   | Change the working directory with CWD                                 |
| P   | Display the current directory with PWD                                |
| H   | Display brief help                                                    |
| Q   | Send QUIT, close the TCP connection, and return to NASSYS             |

For G, the client asks for the remote file name and the hexadecimal start address. It then displays the first free address and the number of bytes received. The selected range must not overwrite the FTP client at 5000h..5EFAh.

The directory display omits permissions, link count, owner, and group. After every twelve entries, -- MORE -- waits for a key press.

For U, the client asks for the remote file name, start address, and length. The address and length are hexadecimal and require no prefix.

After a download, CONTROL can compare memory byte for byte against the original PC file without an intermediate file. Example for a download to 6000h:

```
python -u nascom_control.py --bind 192.168.4.123 `
```

```
--compare-file FTP.bin --load-address 6000h
```

FTP must first be terminated with Q, and CONTROL must then be restarted with E 4000.

### **Transport**

Socket 0 holds the FTP control connection. For LIST, RETR, and STOR, the client requests a port with PASV and connects socket 1. The IP address in the PASV response is deliberately ignored; the previously entered server IP and the reported PASV port are used as the data target. This also works with servers that return an unsuitable private address in their PASV response.

At startup, the client closes sockets 0 and 1 as a precaution. This allows FTP.NAS to be restarted after a Z80 reset or aborted transfer without an old FPGA TCP state blocking the connection attempt.

The TCP core advertises a receive window of 128 bytes. After each block, the window is closed until the Z80 has consumed the data and is then reopened with an ACK. This limits throughput but prevents a fast FTP server from overflowing the small ABI buffer.

### **Firewall and passive data ports**

FTP uses two separate TCP connections:

- The control channel connects to the port entered at program startup, normally TCP port 21.
- For D, G, and U, the client subsequently requests a second, server-selected data port with PASV.

A local FTP server must therefore permit inbound traffic to both the control port and the passive data ports. For FileZilla Server, a small fixed port range such as 50000 through 50100 is recommended:

1. In FileZilla Server, open Protocol settings -> FTP and FTP over TLS (FTPS) -> Passive mode and enable Use custom port range.
2. Enter a range such as 50000-50100.
3. Leave Use local IP for local connections enabled.
4. Permit inbound TCP port 21 and TCP ports 50000-50100 in Windows Firewall.

No router port forwarding is needed for connections within the same local network. For a server behind NAT, the chosen passive range and, if necessary, the control port must also be forwarded to the server.

If login succeeds and the server responds to PASV with 227, but FTP TRANSFER ERROR follows, the control channel is already working. In this case, the second connection most likely cannot reach the passive data port specified by the 227 response. Check the configured passive port range and its firewall rule first.

### **Local test server**

ftptestserver.py provides a reproducible initial test server:

```
python -u ftp_test_server.py --bind 192.168.4.123 --port 21
```

The default directory is ftp-root, relative to the current directory. The server logs control and data connections and supports precisely the commands used by the NASCOM client.

## ADDENDUM 06

### BLS Pascal V1.3 auf dem MEGA65-Nascom2 - erste Schritte

Diese Kurzanleitung gilt für die in diesem Projekt nach \$A000-\$CFFF relokierete BLS-Pascal-Version 1.3. Für V1.3 ist kein eigenes Handbuch bekannt. Die Bedienung wurde deshalb aus dem Handbuch zu V1.2, dem V1.3-Programmabbild und dem Verhalten unserer ROM-Version abgeleitet.

#### 1. Starten

1. Mit F13 die Konfiguration öffnen.
2. Auf Konfigurationsseite 2 BLS PASCAL V1.3 auf ON stellen.
3. Die Änderung mit Enter übernehmen. Der Nascom wird zurückgesetzt.
4. Am NAS-SYS-Prompt eingeben:

```
EA000
```

Das ist der NAS-SYS-Befehl E mit der Startadresse \$A000.

Nach dem Titel erscheint der BLS-Pascal-Prompt:

```
>
```

Der Warmstart liegt bei \$B140 und kann nach einem QUIT mit EB140 aufgerufen werden. Nach einem Reset sollte sicherheitshalber der Kaltstart EA000 verwendet werden.

#### 2. Das erste Programm

Am >-Prompt E und Enter drücken. Damit wird der Vollbildeditor geöffnet. Den folgenden Text eingeben:

```
BEGIN
  WRITELN('HALLO VOM NASCOM 2');
END.
```

Wichtig:

- Befehlswörter dürfen groß oder klein geschrieben werden.
- Text steht in einfachen Anführungszeichen. Das dafür benötigte ASCII-Apostroph ' wird auf der emulierten Nascom-Tastatur mit Shift+7 eingegeben. Der Nascom-Zeichensatz stellt dieses Zeichen auf dem Bildschirm ähnlich wie ´ dar; diese Darstellung ist hier normal.
- Anweisungen werden mit ; getrennt.
- Das Hauptprogramm endet mit END. - einschließlich Punkt.

Mit Ctrl+X den Editor verlassen. Danach steht wieder der >-Prompt da.

Zum Übersetzen eingeben:

```
C
```

Bei Erfolg erscheint ungefähr:

```
Compiling OK
Text: ...
Code ...
```

Die angezeigte Code-Adresse ist der tatsächliche Einsprungpunkt. Zwischen Quelltext und Maschinencode liegt intern ein nicht mitgezähltes CR-Trennbyte.

Anschließend das Programm starten:

```
R
```

R kompiliert automatisch, falls noch kein gültiger Objektcode vorhanden ist. Nach dem Programmende kehrt BLS Pascal zum >-Prompt zurück.

Der kürzeste Arbeitszyklus ist somit:

```
E      Editor öffnen
Ctrl+X Editor verlassen
C      Kompilieren
R      Ausführen
```

### 3. Beispiel mit Eingabe und Variable

```
VAR n: INTEGER;
BEGIN
  WRITE('ZAHL? ');
  READLN(n);
  WRITELN('DOPPELT: ',n*2);
END.
```

WRITE bleibt in derselben Ausgabezeile, WRITELN schließt die Zeile ab. READLN liest eine Eingabe und wartet auf Enter.

Bei der numerischen Eingabe behandelt V1.3 das Ende des Zahlenfeldes und den Abschluss der Eingabezeile getrennt. Falls das Programm nach 7 und dem ersten Enter noch wartet, einfach ein zweites Mal Enter drücken. Der getestete Ablauf liefert danach DOPPELT: 14 und kehrt zum >-Prompt zurück.

### 4. Die wichtigsten Befehle am >-Prompt

Die Befehle dürfen - mit Ausnahme von ZAP - auf den ersten Buchstaben abgekürzt werden.

| Befehl      | Kurz   | Wirkung                                               |
|-------------|--------|-------------------------------------------------------|
| EDIT        | E      | Vollbildeditor öffnen                                 |
| COMPILE     | C      | Quelltext in Z80-Objektcode übersetzen                |
| RUN         | R      | Objektcode ausführen; bei Bedarf vorher kompilieren   |
| MEMORY      | M      | Adressen und Größe von Quell- und Objektcode anzeigen |
| FIND nnnn   | F nnnn | Zu einem Objektcode-Offset die Quellstelle suchen     |
| FIND        | F      | Quellstelle des letzten Laufzeitfehlers suchen        |
| SAVE name   | S name | Quelltext im NAS-SYS-Kassettenformat speichern        |
| LOAD name   | L name | Quelltext im NAS-SYS-Kassettenformat laden            |
| VERIFY name | V name | Gespeicherten Quelltext überprüfen                    |
| QUIT        | Q      | Zu NAS-SYS zurückkehren                               |
| ZAP         | -      | Quelltext und Objektcode löschen                      |

Vorsicht: ZAP wird vollständig ausgeschrieben und löscht den aktuellen Quelltext sowie den Objektcode ohne normalen Rückweg.

SAVE, LOAD und VERIFY meinen das historische NAS-SYS-Kassettenformat. Für Dateien auf der SD-Karte sind stattdessen F9 und F11 vorgesehen:

1. Mit F9 die Speicherseite öffnen. Bei aktivem BLS-Pascal-ROM und vorhandenem Quelltext wird automatisch BLS PASCAL SOURCE gewählt.
2. Einen Dateinamen eingeben. Der Core verwendet automatisch die Endung .PAS und speichert den Quelltextbereich ab \$1000 bis zum aktuellen Textende.
3. Zum Laden F11 öffnen und die .PAS-Datei auswählen. .PAS-Dateien werden im Browser nur angezeigt, wenn das BLS-Pascal-ROM aktiv ist.
4. Nach dem Laden setzt der Core Textende, Editorposition und obere Editoranzeige passend zurück. Der alte Objektcode wird verworfen; deshalb das Programm anschließend mit C neu kompilieren.

Das Dateiformat enthält weiterhin NASCOM-Adresszeilen und lässt sich daher auch außerhalb des Cores untersuchen. Die eigene Endung .PAS verhindert, dass F11 eine gewöhnliche .NAS-Datei irrtümlich als Pascal-Quelltext behandelt.

### 5. Grundlegende Editorbedienung

Normale Zeichen werden an der Cursorposition eingefügt. Eine Quellzeile darf bis zu 80 Zeichen lang sein; das 48-Zeichen-Nascom-Bild ist nur ein bewegliches Fenster auf diese Zeile.

| Taste              | Wirkung im Editor             |
|--------------------|-------------------------------|
| Pfeil rechts/links | Cursor um ein Zeichen bewegen |

| Taste              | Wirkung im Editor                                |
|--------------------|--------------------------------------------------|
| Pfeil hoch/runter  | Cursor um eine Zeile bewegen                     |
| Enter              | Nach unten gehen und eine Zeile einfügen         |
| Backspace          | Nach links gehen und die Position löschen        |
| Shift+Pfeil rechts | Leerzeichen an der Cursorposition einfügen       |
| Shift+Pfeil links  | Zeichen an der Cursorposition löschen            |
| Shift+Pfeil runter | Leerzeile einfügen                               |
| Shift+Pfeil hoch   | Aktuelle Zeile löschen                           |
| Ctrl+B             | Zum Anfang des Quelltexts                        |
| Ctrl+E             | Zum Ende des Quelltexts                          |
| Ctrl+N             | 14 Zeilen nach unten                             |
| Ctrl+O             | 14 Zeilen nach oben                              |
| Ctrl+F             | Text suchen; Suchtext eingeben und Enter drücken |
| Ctrl+C             | Suche mit demselben Suchtext fortsetzen          |
| Ctrl+X             | Editor verlassen und zum >-Prompt zurückkehren   |

Esc löscht im Editor die aktuelle Zeile. Das unterscheidet sich vom BLS-Kommandomodus, wo Esc nur die gerade eingegebene Befehlszeile löscht.

## **6. Compiler- und Laufzeitfehler**

Bei einem Compilerfehler zeigt BLS ungefähr Folgendes:

```
Compilation error 64 Press <SPACE>
```

Nach Druck auf die Leertaste öffnet BLS den Editor und stellt den Cursor an oder unmittelbar hinter die verdächtige Stelle.

Ein Laufzeitfehler sieht ungefähr so aus:

```
Runtime error xx at $nnnn
```

nnnn ist ein Offset innerhalb des erzeugten Objektcodes, keine absolute Adresse. Am >-Prompt genügt anschließend:

```
F
```

BLS kompiliert den Quelltext im Suchmodus und positioniert den Editor bei der zum letzten Laufzeitfehler gehörenden Anweisung.

## **7. Speicherhinweis für unsere \$A000-ROM-Version**

Mit M lässt sich die aktuelle Speicheraufteilung anzeigen. BLS Pascal legt Quelltext und Objektcode zusammenhängend im RAM unterhalb des ROMs ab.

Bei unserer Belegung beginnt das BLS-ROM bei \$A000. Der automatische RAM-Test trifft dort auf ROM und betrachtet deshalb normalerweise \$A000 als obere Grenze des zusammenhängenden Arbeitsspeichers. In unserer angepassten V1.3 beginnt der Pascal-Quelltext bei \$1000 statt wie im wiedergefundenen RAM-Abbild bei \$4000. Damit steht BLS Pascal der zusammenhängende Bereich \$1000-\$9FFF für Quelltext, erzeugten Objektcode und seine wachsenden Arbeitsbereiche zur Verfügung.

Damit Compiler-Arbeitsbereich und Stack nicht beide an \$A000 beginnen und ineinanderlaufen, verwendet unsere ROM-Fassung für Compiler- und Programmlaufzeit den Stackzeiger \$0000. Der Stack wächst dadurch ab \$FFFF im oberen RAM nach unten. Normalerweise steht ihm \$E000-\$FFFF zur Verfügung; \$D000-\$DFFF bleibt bei aktiver FDD-Konfiguration NAS-DOS vorbehalten. Der Stack darf diese Grenze nicht erreichen.

Das zusätzliche RAM bei \$E000-\$FFFF wird somit für den Stack genutzt. Es kann wegen der ROM-Lücke \$A000-\$CFFF nicht als zusammenhängender Quell-/Objektspeicher verwendet werden.

## **8. Nützliche Sprachelemente für den Anfang**

V1.2 und das V1.3-Abbild enthalten unter anderem:

- Typen INTEGER, REAL, BOOLEAN und STRING;

- IF ... THEN ... ELSE, CASE, WHILE, REPEAT ... UNTIL und FOR ... TO/DOWNT0;
- Prozeduren und Funktionen;
- READ, READLN, WRITE und WRITELN;
- Nascom-Erweiterungen wie SCREEN, PLOT, INP und OUT.

Für die ersten Versuche empfiehlt es sich, bei einfachen Integer-Programmen, WRITE/WRITELN und kurzen Zeilen zu bleiben. Danach können wir die Grafik-, Port-I/O- und Datei-Erweiterungen gezielt untersuchen.

#### **Quellen und Versionshinweis**

- Historisches BLS-Pascal-V1.2-Handbuch:  
<https://nascomhomepage.com.mirror.jloh.de/pdf/blspas.pdf>
- Nascom Home Page: Wiederfund von V1.3: <https://nascomhomepage.com.mirror.jloh.de/>
- Lokales V1.3-ROM und Disassemblerlisting in diesem Projekt

Das V1.2-Handbuch beschreibt den Programmstart und einige Adressen seiner Band- und ursprünglichen ROM-Version. Diese historischen Adressen gelten nicht für unsere Relokation. Für dieses Projekt sind \$A000 (Kaltstart) und \$B140 (Warmstart) maßgeblich.

## NASCOM Control Agent - GUI Interface

## Overview

- Agent file: CONTROL.NAS
- Load address: 4000h
- NASSYS start command: E 4000
- Memory occupied by the agent: 4000h through 46DCh, inclusive
- Agent target: IPv4 address entered at startup, TCP port 2323
- FPGA TCP socket used: 2
- Maximum payload per protocol frame: 120 bytes
- Protocol identifier: N2C, version 1

The default address and fixed port are stored in the six-byte TARGET field in CONTROL.asm: four IPv4 bytes followed by the TCP port in network byte order. Startup input replaces the first four bytes; the port bytes remain 09 13 for port 2323.

1. Start a TCP listener on the desired local IPv4 address and port 2323.
2. Start CONTROL.NAS with E 4000 and enter the listener address; empty input selects 192.168.4.123.
3. Accept the connection from the NASCOM.
4. Wait approximately one second after accept().
5. Send a PING with sequence number 0.
6. Validate the response. The expected payload is 4E 32 43 01, meaning "N2C" and protocol version 1.
7. Thereafter, send exactly one request at a time and wait for its complete response.

A useful GUI state sequence is:

| |

+---- ERROR -----+

QUIT, a broken TCP connection, or an error in executed Z80 code terminates

the session. The GUI must then listen again and the agent must be restarted.

### **TCP is a byte stream**

A call to `recv()` does not necessarily return exactly one protocol frame. A frame can be split across several TCP segments, and several frames can arrive together. The GUI must buffer bytes and use the magic value, length field, and checksum to assemble complete frames.

The recommended implementation uses stop-and-wait operation:

- allow at most one outstanding request;
- read the complete response;
- validate command and sequence number;
- only then send the next request.

### **Request frame**

| Offset | Size | Meaning                   |
|--------|------|---------------------------|
| 00h    | 2    | Magic: 4E 32 (ASCII "N2") |
| 02h    | 1    | CMD                       |
| 03h    | 1    | SEQ                       |
| 04h    | 1    | ARG_H                     |
| 05h    | 1    | ARG_L                     |
| 06h    | 1    | LEN                       |
| 07h    | LEN  | optional payload          |
| ...    | 1    | XOR checksum              |

Short form:

4E 32 CMD SEQ ARG\_H ARG\_L LEN [PAYLOAD] XOR

ARGH:ARGL is a 16-bit value in network byte order, high byte first. For memory commands it is the Z80 address. For port commands, the low byte ARG\_L is used as the 8-bit port number.

### **Response frame**

| Offset | Size | Meaning                      |
|--------|------|------------------------------|
| 00h    | 2    | Magic: 32 4E (ASCII "2N")    |
| 02h    | 1    | Response command: CMD OR 80h |
| 03h    | 1    | SEQ from the request         |
| 04h    | 1    | STATUS                       |
| 05h    | 1    | response payload LEN         |
| 06h    | LEN  | optional payload             |
| ...    | 1    | XOR checksum                 |

Short form:

32 4E (CMD|80h) SEQ STATUS LEN [PAYLOAD] XOR

The GUI must verify that the response command equals CMD | 80h and that the sequence number matches the outstanding request.

### **XOR checksum**

The two magic bytes are not included in the checksum. The XOR checksum is the

XOR of every byte from CMD through the final payload byte.

```
checksum = CMD XOR SEQ XOR ... XOR final payload byte
```

A convenient validation of a received frame is:

```
XOR(CMD through and including transmitted checksum byte) == 00h
```

Python example:

```
def xor_checksum(data: bytes) -> int:
    value = 0
    for byte in data:
        value ^= byte
    return value
```

### Sequence number

SEQ is a freely assigned 8-bit counter maintained by the GUI. Start at 0 and increment modulo 256 after each request:

```
00, 01, 02, ... FE, FF, 00, ...
```

The sequence number detects delayed responses and responses that do not belong to the outstanding request. Nevertheless, only one request should be outstanding at any time.

### Commands

| Cmd | Name         | ARG          | Request LEN/payload                           | Success response                                     |
|-----|--------------|--------------|-----------------------------------------------|------------------------------------------------------|
| 01h | PING         | 0000h        | LEN=0   4 bytes: 4E 32 43 01 (N2C, version 1) |                                                      |
| 10h | MEMORY_READ  | Z80 address  | LEN=1..120, no request payload                | exactly LEN memory bytes                             |
| 11h | MEMORY_WRITE | Z80 address  | LEN=1..120, exactly LEN payload bytes         | empty response                                       |
| 20h | PORTIN       | port in ARGL | LEN=0                                         | one byte read from the port                          |
| 21h | PORTOUT      | port in ARGL | LEN=1, one payload byte                       | empty response                                       |
| 30h | EXEC         | Z80 address  | LEN=0                                         | empty response, followed by execution at the address |
| 7Fh | QUIT         | 0000h        | LEN=0                                         | empty response, followed by connection shutdown      |

### PING (01h)

PING serves as both the handshake and a connection test.

Request with SEQ=00h:

```
4E 32 01 00 00 00 00 01
```

Response:

```
32 4E 81 00 00 04 4E 32 43 01 BB
```

### Read memory (10h)

ARGH:ARGL contains the start address. LEN specifies the requested number of response bytes and must be between 1 and 120. Despite the nonzero length field, the request contains no payload: for MEMORY\_READ, LEN describes the response length.

Example: read 16 bytes at address 0000h, SEQ=02h:

```
Request: 4E 32 10 02 00 00 10 02
```

```
Response: 32 4E 90 02 00 10 [16 memory bytes] XOR
```

The GUI must split larger reads into blocks of at most 120 bytes. A block should not cross the FFFFh -> 0000h boundary.

### **Write memory (11h)**

ARGH:ARGL contains the start address. LEN must be between 1 and 120 and must exactly match the number of following payload bytes.

Example: write 01 02 03 to 4700h, SEQ=5Ah:

Body: 11 5A 47 00 03 01 02 03

Request: 4E 32 11 5A 47 00 03 01 02 03 0F

Response: 32 4E 91 5A 00 00 CB

The agent itself does not protect its memory range. A GUI should block writes to 4000h..46DCh by default. This protection in the supplied Python program is a host-side feature, not part of the binary protocol.

### **Read port (20h)**

The port number is stored in ARGL; ARGH should be 0. LEN is 0. The response contains exactly one byte.

CMD=20h, ARG=0002h -> IN A,(02h)

### **Write port (21h)**

The port number is stored in ARG\_L. LEN is 1 and the only payload byte is the value to output.

CMD=21h, ARG=0002h, LEN=1, PAYLOAD=40h -> OUT (02h),40h

### **Execute Z80 code (30h)**

The agent first sends a successful response and then calls the address as a subroutine. The executed code must return to the agent with RET for further GUI commands to be processed.

The response confirms only that the call was accepted. It does not confirm that the called code has already completed. The GUI should not send further requests while external code is running.

### **End session (7Fh)**

The agent sends an empty success response, closes the TCP socket, and returns to NASSYS. After receiving the response, the GUI should expect an orderly TCP shutdown.

### **Status codes**

|          |                                              |  |
|----------|----------------------------------------------|--|
| STATUS   | Meaning                                      |  |
| ---: --- |                                              |  |
| 00h      | success                                      |  |
| 01h      | unknown command                              |  |
| 02h      | invalid length or invalid command parameters |  |
| 03h      | incorrect XOR checksum                       |  |

An error response currently has a payload length of 0. The GUI should still handle LEN generically and consume that many bytes so future protocol extensions remain possible.

### **Robust response parser**

Recommended procedure:

1. Search the receive buffer for 32 4E.
2. Read the next four header bytes completely: command, sequence, status, length.
3. Read exactly LEN payload bytes.
4. Read exactly one checksum byte.
5. Validate the XOR of response header, payload, and checksum.
6. Compare response command and sequence number with the outstanding request.
7. Display a GUI error when STATUS != 0.

Pseudocode:

```
def receive_exact(sock, count):
    result = bytearray()
    while len(result) < count:
        chunk = sock.recv(count - len(result))
        if not chunk:
            raise ConnectionError("NASCOM connection closed")
        result.extend(chunk)
    return bytes(result)

def receive_response(sock):
    sync = bytearray()
    while bytes(sync[-2:]) != b"2N":
        sync.extend(receive_exact(sock, 1))
    if len(sync) > 2:
        del sync[:-2]
    header = receive_exact(sock, 4)
    response_command, sequence, status, length = header
    payload = receive_exact(sock, length)
    checksum = receive_exact(sock, 1)[0]
    if xor_checksum(header + payload + bytes([checksum])) != 0:
        raise ValueError("invalid response checksum")
    return response_command, sequence, status, payload
```

### **Block transfers**

For files and larger memory ranges:

- use no more than 120 bytes per frame;
- advance the address by the block length after each block;
- do not read or write across FFFFh within one block;
- update progress only after receiving a successful response;
- do not silently skip the current block after an error.

Example of a safe block size:

```
count = min(remaining, 120, 0x10000 - address)
```

### **GUI recommendations**

Useful controls include:

- connection status and NASCOM source address;
- Connect/Listen and Disconnect/QUIT;
- hexadecimal/ASCII memory display;
- load and save memory;
- edit memory and write it back;
- port number, port value, IN, and OUT;
- entry address and EXEC;
- protocol log containing CMD, SEQ, STATUS, and length;
- progress indication for block transfers.

While in the BUSY state, disable all controls that would issue another request. After losing the connection, return the GUI to LISTENING.

### **Reference implementation**

The complete tested host implementation is in `nascom_control.py`. The following functions are particularly relevant to a GUI:

- `xor_checksum()`
- `recvexact()` and `recvresponse()`
- `NascomController.transact()`
- `readmemory()` and `writememory()` for block transfers
- `readport()`, `writeport()`, `execute()`, and `quit()`

A binary image can also be loaded non-interactively, read back completely, and then finished with a clean QUIT:

```
python -u nascom_control.py --bind 192.168.4.123 `
    --load-file FTP.bin --load-address 5000h
```

With `--compare-file FILE --load-address ADDRESS`, the same range is only read and compared byte for byte with a PC file; the tool then terminates the agent cleanly as well.

The hardware smoke test covers PING, ROM reads, reversible RAM writes, port reads, QUIT, and the return to NASSYS.