# SD81 BOOSTER

## User manual




Revised edition · English
23 / 10 / 2026
SD · WiFi · Firmware · FPGA · ROM

# Contents

About this edition ..... 3
Quick Start Guide ..... 3
1. Introduction ..... 4
2. Hardware Description ..... 4
3. Box Contents ..... 7
4. Installation ..... 7
5. microSD Card Preparation ..... 8
6. Getting Started ..... 9
7. Loading and Saving from SD ..... 9
8. File and Directory Management ..... 12
9. Additional Features ..... 13
10. Sound ..... 21
11. Memory Management ..... 24
12. Advanced BASIC Extensions ..... 27
13. Program Examples ..... 30
14. Error Codes ..... 33
15. For Programmers ..... 34
16. Troubleshooting ..... 42
17. Updating through microSD: firmware, FPGA, ROM and WiFi ..... 45
18. Glossary ..... 47
19. Firmware Version History ..... 48
20. References ..... 48
Appendix A — Complete PLAY Command Reference ..... 49
Appendix B — PEG Effects Generator Reference ..... 51
Appendix C — Speech Synthesiser Dictionary ..... 52
Appendix D — Memory Paging System ..... 53
Appendix E — Chroma81 Interface Port (7FEFh) ..... 54
Appendix F — Superfast and Spectrum Modes ..... 56
Appendix G — Audio Technical Reference: AY chip, VGM and allophones ..... 63
Appendix H — Hardware sprites ..... 65
Appendix I — CP/M: an alternative operating system ..... 67
Appendix J — The file explorer ..... 73
Appendix K — The EightyOne-CrossPlatform emulator ..... 77
Appendix L — Hardware debugger ..... 85
Appendix M — Networking for programmers: MQTT and turn-based games ..... 89
Review sources ..... 92

# About this edition

Documentary review of Alejandro Valero's (wilco2009) repository. Retains the source manuals' chapters and appendices, with revised update instructions, NTP behavior, references and contents. This is not a hardware update test.

**Inspected revision:** `39b367a7e980157512b4c1928d10e97aedf1cfad` (20/10/2026).

**Edition date:** 23/10/2026.

Hardware design and MCU firmware: Alejandro Valero (wilco2009). Z80 code / modified ROM: Pedro Gimeno (pgimeno). Source text and illustrations: SD81 Booster project.

Repository and sources · Releases

Section 17 is the expanded update guide. Before using binaries from a later release, read its notes: filenames or requirements may change.

# Quick Start Guide

If you have just opened the box and want to get started as quickly as possible, follow these five steps:

**1. Prepare the microSD card**

- Format a microSD card in FAT32.
- Copy the SYS folder and all its contents to the root of the card. Without this folder the interface will not boot.
- Copy your .P files to the card, organised in folders if you wish.

**2. Connect the interface**

- Turn off the ZX81.
- Connect the SD81 Booster to the rear expansion port of the ZX81.
- Insert the microSD card into the interface slot.

**3. Power on and check**

- Turn on the ZX81. It should boot normally showing the K cursor.
- The STAT LED on the top panel should light up solid green.

**4. Load your first program**

> LOAD FAST "NAME"

Replace NAME with the filename without the .P extension.

**5. Want to explore what's on the SD?**

> LOAD *DIR

| INFO | For more details on any of these steps, refer to the corresponding sections of the manual. |
|---|---|

# 1. Introduction

The SD81 Booster is an open hardware expansion interface for the Sinclair ZX81 that considerably extends the computer’s original capabilities. Its main features include:

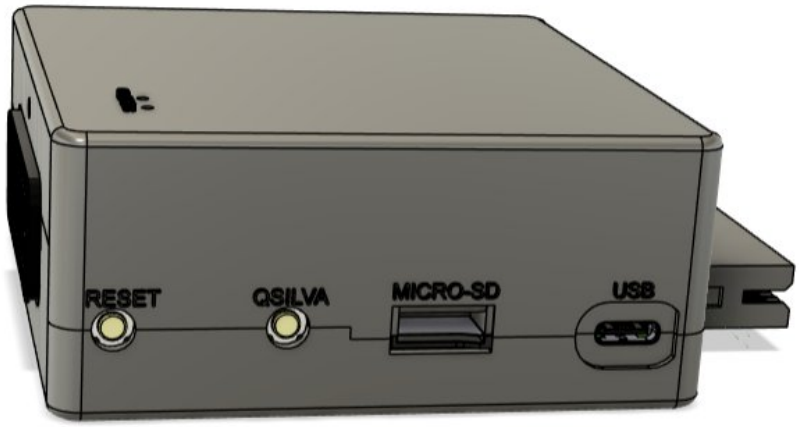
- Loading and saving from microSD card in .P format, the same used by the most popular emulators.
- Up to 512 KB of RAM via a memory mapper in 8 KB blocks.
- AY-3-8910/12 sound chip emulation with support for the PLAY command.
- VGM file player running in the background.
- Speech synthesis with audio samples stored in the microcontroller’s internal memory.
- Up to 128 user-definable characters, with compatibility with the QuickSilva interface character definition mode.
- Chroma81 interface compatibility, providing RGB colour video output for the ZX81.
- Full compatibility with the original tape routines and the ZX Printer.

# 2. Hardware Description

The SD81 Booster is a compact enclosure that connects to the ZX81’s rear expansion port. On its exterior you will find all the connectors and controls needed to make use of its features.

| INFO | **The images in this section are provisional 3D design renders and will be replaced with photographs of the final product before launch.** |
|---|---|

## 2.1 Right Side Panel



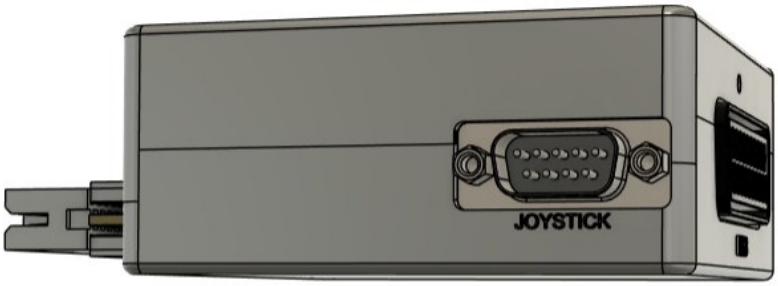

Right side view: RESET, QSILVA, MICRO-SD and USB

The right side panel presents, from left to right:

- RESET button: resets the entire system (ZX81 + interface).

- QSILVA button: toggles between the QuickSilva interface character set and the standard ZX81 ROM character set. Each press switches between one and the other. Held for 1 to 3 seconds, it pauses the program and shows the debugger; for more than 3 seconds, it saves a snapshot (see Appendix L).
- MICRO-SD slot: insert the microSD card here with your programs, ROMs and system data.
- USB-C port: has two uses. As a firmware update port in case USB recovery is needed (see section 17). As a debug console: when connected to a computer a serial port associated with the CH340 chip appears (may require drivers). With a serial terminal program it is possible to monitor the interface's status and error messages in real time.

## 2.2 Left Side Panel and Rear Panel




Left side view: JOYSTICK DB9 connector

- JOYSTICK connector (DB9): joystick port compatible with standard 9-pin joysticks (Atari/Commodore type). The mapping of joystick buttons to ZX81 keys is fully programmable via the LOAD *JOY command (see section 9.6).




Rear view: RGB SCART connector

- RGB connector (SCART): RGB colour video output compatible with the Chroma81 interface. Allows the ZX81 to be connected to monitors and televisions with SCART input for high quality colour image.

## 2.3 Top Panel — Status LEDs




Top view: STAT and SD LEDs

The top panel incorporates two indicator lights:

- STAT LED (status): indicates the general state of the interface through different colours and blink patterns (see table in section 2.4).
- SD LED (card access): blinks during read or write operations on the microSD card.

## 2.4 STAT LED Status Table

The STAT LED indicates the state of the interface through colour and blink combinations, organised in three phases:

| Colour / Pattern | Meaning |
|---|---|
| **Firmware update** | |
| **Blue/Red blink** | Error initialising SD card |
| **Solid yellow** | Updating firmware |
| **White/Red blink** | Update error |
| **Boot** | |
| **Red blinking** | Initialising serial port |
| **Orange blinking** | Waiting for FPGA |
| **Blue/Red blink** | Error initialising SD card |
| **Orange/Red blink** | Error writing ROM to RAM |
| **Yellow blink** | Initialising RTC |
| **Solid green** | System initialised successfully |

| Colour / Pattern | Meaning |
|---|---|
| **Operation** | |
| **Solid green** | Interface ready, Quick Silva mode off |
| **Solid cyan** | Interface ready, Quick Silva mode on |

# 3. Box Contents

When you open the SD81 Booster box you will find:

- 1× SD81 Booster interface
- 1× microSD card (pre-formatted)
- 1× CR2032 button battery (pre-installed in the interface)
- This user manual

| NOTE | If any of these items are missing or damaged, contact the seller before connecting the interface. |
|---|---|

# 4. Installation

## 4.1 Before Connecting the Interface

- Make sure the ZX81 is powered off before connecting or disconnecting the interface.
- The SD81 Booster connects to the ZX81’s rear expansion port.
- The interface is not compatible with devices that replace the ZX81’s internal ROM.

## 4.2 Connection

1. Turn off the ZX81.
2. Align the SD81 Booster connector with the ZX81’s rear expansion port. Make sure the pins are correctly aligned.
3. Push gently until the connector is fully inserted. Do not force the connection.
4. Insert the microSD card into the interface slot (see section 5).
5. Turn on the ZX81.

| NOTE | Connecting or disconnecting the interface while the ZX81 is powered on may damage both the interface and the computer. |
|---|---|

## 4.3 Installation Check

When powering on the ZX81 with the SD81 Booster correctly installed, the computer should boot normally showing the usual K cursor. The interface does not modify the system boot process.

To verify the interface is working, type the following command in BASIC and press ENTER:

```
LOAD *VER
```

| INFO | The asterisk * is obtained by pressing SHIFT + B. The interface will display the installed firmware version. |
|---|---|

If the computer freezes or does not boot correctly, disconnect the interface and make sure the connector is properly aligned.

# 5. microSD Card Preparation

## 5.1 Card Format

The SD81 Booster requires a microSD card formatted in FAT32. The card included in the box is already correctly formatted and prepared.

If you use your own card, follow these steps:

1. Format the card in FAT32: on Windows, right-click on the card → Format → FAT32. On macOS or Linux, use a disk formatting tool and select FAT32.
2. Copy the SYS folder and all its contents (included in the interface software package) to the root of the SD card. This folder contains essential files for the interface to operate.

| NOTE | The interface does not support exFAT or NTFS format. The SYS folder is essential: it contains the ROM files needed for the interface to boot. Without it, the interface will not work. |
|---|---|

## 5.2 Permitted Characters in Filenames

Due to the ZX81 keyboard limitations, only the following characters may be used in file and folder names:

- Letters: A to Z (always uppercase when saving)
- Numbers: 0 to 9
- Symbols: . , ; $ ( ) = + -
- The / character is used as a directory separator and cannot be used in filenames.

| INFO | Avoid ending a filename with a space or a period, as some operating systems may have trouble reading that file from a computer. |
|---|---|

## 5.3 Recommended Folder Structure

The interface will search the SD card root by default. You can organise your programs in subfolders. The following structure is recommended:

```
/
├── AUTOEXEC.P ← Program that loads at boot
├── JUEGOS/
│   ├── MANIC.P
│   └── PACMAN.P
├── DEMOS/
└── SYS/ ← System folder (mandatory, do not modify)
```

### 5.4 The AUTOEXEC Program

If a file called AUTOEXEC.P exists in the SD root, it will be automatically loaded and executed when the RUN command is typed on a ZX81 with no program loaded. This is useful for creating custom startup menus.

# 6. Getting Started

### 6.1 Load Modes: SD or Tape

By default, the BASIC LOAD and SAVE commands work with tape, exactly as on a ZX81 without an interface. This ensures full compatibility with original tape software.

To use the SD card, you have two options:

**Option A — Activate SD mode permanently:** Type in BASIC:

> LOAD FAST

| INFO | **The word FAST is obtained with SHIFT + F, not by typing the letters one by one. From that point on all LOAD and SAVE commands will use the SD by default. If you need to load from tape while in SD mode, use LOAD SLOW “name” to force audio loading for that specific operation.** |
|---|---|

To return to tape mode by default:

> LOAD SLOW

| INFO | **SLOW is obtained with SHIFT + D. From that point on all LOAD and SAVE commands will use tape by default, and FAST must be explicitly added to load from SD.** |
|---|---|

**Option B — Load from SD without changing mode:** Add FAST directly to the load command (see section 7).

### 6.2 Loading Your First Program from SD

Once SD mode is active (or using LOAD FAST), loading a program is straightforward. Replace NAME with the name of your file (without the .P extension):

> LOAD FAST “NAME”

Press ENTER. The program will load in a few moments and will start automatically if it has auto-run.

| INFO | **If you do not remember the exact filename, you can view the SD contents with the command LOAD *DIR (see section 8).** |
|---|---|

# 7. Loading and Saving from SD

### 7.1 Load a Program

To load a program from the SD card:

> LOAD FAST “NAME”

The interface will first try the exact filename. If not found, it will automatically try adding the .P extension.

To load from a subfolder:

```
LOAD FAST “JUEGOS/PACMAN”
```

To load and run from a specific line number:

```
LOAD FAST “NAME” THEN GOTO 100
```

To load without auto-running:

```
LOAD FAST “NAME” THEN STOP
```

| INFO | **THEN, GOTO y STOP are ZX81 BASIC tokens, not written letter by letter.** |
|---|---|

## 7.2 Save a Program

To save a program to the SD card:

```
SAVE FAST “NAME”
```

The interface always appends the .P extension to the saved file.

| NOTE | **If a file with that name already exists, it will be overwritten without warning.** |
|---|---|

## 7.3 Load and Save Memory Blocks (Machine Code)

To load a block of data to a specific memory address:

```
LOAD FAST “NAME” CODE 30000
```

| INFO | **Unlike the ZX Spectrum, the address after CODE is mandatory. Files on SD do not store the load address in a header.** |
|---|---|

| NOTE | **The FAST token is mandatory for CODE to work correctly. If LOAD “NAME” CODE 30000 is used without FAST (even with SD mode active), the file will be loaded at the default address 16393 (4009h), ignoring the specified address. This is a firmware compatibility limitation.** |
|---|---|

To save a memory block to SD:

```
SAVE FAST “NAME” CODE 30000,2048
```

Where 30000 is the start address and 2048 is the length in bytes.

## 7.4 Always Load from Tape (Regardless of Mode)

If SD mode is active but you want to load a specific file from tape:

```
LOAD SLOW “NAME”
```

## 7.5 Game Compatibility Notes

Some games require special initialisation before loading. The most common cases are:

- **LOAD *128C** before loading: the game uses user-definable characters mode.
- **LOAD FAST** before loading: some games have multiple files. Running LOAD FAST without a name activates SD mode for all subsequent LOAD and SAVE commands, allowing the game to load its additional files automatically.

- **LOAD *ROMLOCK** before loading: some old programs write to addresses in the ROM area on their own (for example to probe for RAM), unaware that the interface uses some of them as configuration ports (2041-2062, 2090-2098 and the sprites, 2100-2128). With LOAD *ROMLOCK those writes are ignored, as with a real ROM; LOAD *ROMLOCK STOP restores normal behaviour. It is off by default and can only be changed with the command, never by a memory write.

## 7.6 Recognised File Formats

The LOAD FAST command automatically detects the file type by extension and behaves differently accordingly:

| Extension | Behaviour |
|---|---|
| **.P** | Standard ZX81 BASIC program. The interface calculates the real program size from the system variables and discards trailing bytes at the end of the file. |
| **.81** | Same as .P. |
| **.P81** | Multi-program format. The interface skips the embedded filename before reading the data. |
| **.ROM** | ROM file. The interface loads the content at address 0 of the address space and resets the system. Control does not return to BASIC. |
| **.WAV** | Uncompressed audio file (PCM). The interface plays it directly instead of loading it into memory. |
| **Others** | The file is loaded entirely into memory as-is, without any processing. |

| NOTE | **Loading a file with the .ROM extension via LOAD FAST causes an immediate system reset. Make sure the file contains a valid ROM before loading it, as a corrupted file could leave the system in an unrecoverable state until it is rebooted with another ROM.** |
|---|---|

## 7.7 Loading EightyOne Snapshots — Z81 Command

The LOAD *Z81 command restores a .Z81 snapshot saved with the EightyOne emulator (see Appendix K): the memory and the full processor state, so the program carries on exactly where it was saved.

**Load a snapshot:**

```
LOAD *Z81 “GAME.Z81”
```

The name must include the .Z81 extension and may include a path, just like LOAD FAST. Besides memory and registers, the NMI state, WRX mode, character generator (64, 128 or 256) and Chroma81 colour (colour RAM and mode) are restored if the snapshot includes them; anything missing from the file is left as it was. If the file doesn’t exist or isn’t a valid snapshot, the command ends with an error. Snapshots are loaded by the hardware debugger (Appendix L): the monitor /SYS/DEBUG.BIN must be on the SD card (it comes in SD Content), and the command cannot be used while the program is stopped in the debugger or while another snapshot is being saved or loaded. In those cases it ends with error P and nothing is loaded.

It is also useful for programs that don’t start properly with the interface connected: save a snapshot in the emulator with the program already running and load it directly, skipping its startup code. The file explorer (Appendix J) loads .Z81 files with this command.

Don’t use LOAD FAST on a .Z81 file: since it isn’t a recognised extension (see 7.6), it would be loaded into memory as-is, without being interpreted.

| NOTE | **EightyOne snapshots are restored as the Z80 sees them (8K to 64K). Those saved by the SD81 Booster itself (see Appendix L) also hold the extended RAM pages and their assignment, the control POKEs, the sound and the interface state, and LOAD *Z81 restores them in full when the debugger is loaded (/SYS/DEBUG.BIN). Without the debugger no snapshot can be loaded (error P).** |
|---|---|

# 8. File and Directory Management

## 8.1 View SD Contents

To list the files in the current directory:

```
LOAD *DIR
```

To list the files in a specific folder:

```
LOAD *DIR “JUEGOS”
```

Wildcards in Unix style can be used:

- * matches any number of characters (including none).
- ? matches exactly one character.

| INFO | *** matches all files regardless of whether they have an extension or not. In contrast, *.* only matches files that contain a dot in their name. To list all files, use *, not *.*.** |
|---|---|

For example, to list only .P files:

```
LOAD *DIR “*.P”
```

| INFO | **If the listing does not fit on screen, ... will appear on the bottom line. Press any key to continue, or SPACE to cancel.** |
|---|---|

## 8.2 Change Directory

To change to a subdirectory:

```
LOAD *CD “JUEGOS”
```

To go back to the root:

```
LOAD *CD “/”
```

View the current directory:

```
LOAD *PWD
```

## 8.3 Create and Delete Folders

To create a subdirectory in the current directory:

```
LOAD *MD “NEWFOLDER”
```

To remove an empty folder:

```
LOAD *RD “EMPTYFOLDER”
```

## 8.4 Delete, Rename and Copy Files

To delete a file:

```
LOAD *DEL “FILE.P”
```

To rename or move a file:

```
LOAD *MV “OLD.P” TO “NEW.P”
```

To copy a file:

> LOAD *CP “SOURCE.P” TO “DEST.P”

| | |
|---|---|
| **INFO** | **TO is a BASIC token (SHIFT + 4), not typed letter by letter.** |
| **INFO** | The date and time of the destination file is not preserved; the copied file will have the date and time of the moment of copying. |

### 8.5 Free Space on SD

> LOAD *FREE

| | |
|---|---|
| **INFO** | **This calculation may take several seconds to complete.** |

### 8.6 T81 Directories (feature in alpha stage)

| | |
|---|---|
| **NOTE** | **This feature is currently in alpha stage. It may contain bugs and its behaviour or interface could change in future versions. Not recommended for production use.** |

The SD81 Booster supports a special file format with the .T81 extension that acts as a container for multiple ZX81 programs, similar to how a ZIP file contains several files. This allows distributing program collections in a single file.

To access the contents of a T81 file, use the LOAD *CD command as if it were a normal directory:

> LOAD *CD “COLECCION.T81”

From that point on, LOAD, LOAD *DIR and LOAD *OPENDIR commands operate on the T81 file contents instead of the FAT filesystem, allowing navigation and loading of programs in the same way as normal files.

To exit the T81 directory and return to the normal filesystem:

> LOAD *CD “/”

| | |
|---|---|
| **INFO** | **Alpha stage limitations: Write operations (LOAD *DEL, LOAD *MD, LOAD *RD, LOAD *MV, LOAD *CP, SAVE) are not supported inside a T81 directory and will return an error.** |

# 9. Additional Features

## 9.1 WAV Audio File Playback

The interface can play uncompressed WAV audio files directly from the SD card simply by loading them with the usual command:

> LOAD FAST “SOUND.WAV”

If the file has the .WAV extension, the interface detects it automatically and plays it instead of trying to load it as a program. No special command is needed.

| | |
|---|---|
| **INFO** | **Only uncompressed WAV audio files (PCM) are supported. Other audio formats are not compatible.** |

## 9.2 Real Time Clock — RTC Command

The SD81 Booster incorporates a real time clock (RTC) with backup battery. The LOAD *RTC command allows checking and setting the time and date from BASIC.

**Read the current date and time:**

> LOAD *RTC

**Store the value in a string variable:**

> LOAD *RTC TO R$

**Set the date and time:**

> LOAD *RTC=“date/time”

Six input formats are supported:

| Format | Example | Description |
|---|---|---|
| YYYY-MM-DD HH:MM:SS.CC | 2025-04-30 18:30:00.00 | Full date and time with hundredths |
| YYYY-MM-DD HH:MM:SS | 2025-04-30 18:30:00 | Full date and time |
| YYYY-MM-DD | 2025-04-30 | Date only |
| HH:MM:SS.CC | 18:30:00.00 | Time only with hundredths |
| HH:MM:SS | 18:30:00 | Time with seconds |
| HH:MM | 18:30 | Time (hours and minutes) |

**Examples:**

> LOAD *RTC=“2025-04-30 18:30:00”
>
> LOAD *RTC=“2025-04-30”
>
> LOAD *RTC=“18:30:00”

**Example in a BASIC program:**

```
10 LOAD *RTC TO R$
20 PRINT “Date and time:”;R$
```

## 9.3 RTC Battery Status — BAT Command

The SD81 Booster incorporates a CR2032 button battery that keeps the clock running when the ZX81 is powered off. This battery comes pre-installed from the factory and has an estimated life of several years under normal use conditions. When it runs out, it can be replaced by any standard CR2032 battery available from electronics shops.

The battery charge level can be checked from BASIC with the LOAD *BAT command.

**Display battery status on screen:**

> LOAD *BAT

**Store the status in a string variable:**

> LOAD *BAT TO B$

| INFO | If the clock frequently loses time when the computer is powered off, check the battery status with this command to see if it needs replacing. |
|---|---|

## 9.4 Extended RAM Mode — RAM48 Command

The LOAD *RAM48 command activates extended RAM mode of 48 KB, which extends the memory available for BASIC programs and data beyond the usual limits.

**Activate extended RAM mode:**

```
LOAD *RAM48
```

**Deactivate to restore standard compatibility:**

```
LOAD *RAM48 STOP
```

| INFO | STOP is the BASIC token, not typed letter by letter. If any program has compatibility issues with RAM48 mode active, deactivate it with LOAD *RAM48 STOP before loading it. |
|---|---|

## 9.5 Text File Display — THEN PRINT Command

The LOAD THEN PRINT command is the equivalent of the MS-DOS TYPE command or Linux cat: it displays the content of a text file directly on the ZX81 screen.

**Display a text file on screen:**

```
LOAD THEN PRINT “FILE”
```

**Send the content to the ZX Printer:**

```
LOAD THEN LPRINT “FILE”
```

**Redirect other command output to the ZX Printer:**

The LOAD LPRINT prefix can also be used with commands that normally display text on screen, to redirect their output directly to the ZX Printer:

```
LOAD LPRINT DIR
LOAD LPRINT DIR “*.P”
LOAD LPRINT FREE
LOAD LPRINT PWD
LOAD LPRINT VER
```

### Built-in Help System

If an asterisk * is prepended to the filename, the interface automatically searches for a file with that name in the /MAN/ folder on the SD and appends the .TXT extension. This allows implementing a help system similar to the Linux man command:

```
LOAD THEN PRINT “*PLAY”
```

This command would look for the file /MAN/PLAY.TXT on the SD and display its content on screen. You can create your own help files in that folder to document your programs or commands.

**Example — display game instructions:**

```
10 LOAD THEN PRINT “*INSTRUCTIONS”
```

This would display the content of /MAN/INSTRUCTIONS.TXT, ideal for showing the instructions of a game or program from within the BASIC program itself.

## 9.6 Programmable Joystick — JOY Command

The SD81 Booster incorporates a DB9 joystick port whose button mapping is fully configurable. The LOAD *JOY command allows assigning a ZX81 key to each direction and the fire button of the joystick.

**Syntax:**

> LOAD *JOY “up/down/left/right/fire”

The configuration string contains exactly five characters, one for each joystick function in this order: up / down / left / right / fire.

**Example:**

> LOAD *JOY “QAOP”

- Up → key Q
- Down → key A
- Left → key O
- Right → key P
- Fire → space key

**Special keys:**

The SHIFT and ENTER keys have no character of their own that can be typed inside the configuration string, so they are represented by two substitute characters: * for SHIFT and < for ENTER.

> LOAD *JOY “QAOP<”

In this example the fire button is assigned to the ENTER key.

| INFO | **Check the controls of each game before configuring the joystick. Many ZX81 games use different key combinations, and with this command you can map them to any standard 9-pin joystick without modifying the software.** |
|---|---|

## 9.7 WiFi Module (optional)

The WiFi module is an optional accessory based on an ESP32-C3 microcontroller that connects to the SD81 Booster and adds a file server accessible over WiFi: it lets you list, upload, download, delete and organise the content of the microSD card from the browser of a phone, tablet or computer, without removing the card from the interface.

| INFO | The WiFi module is not included by default with the SD81 Booster - it is an optional accessory installed separately. If your interface does not have it installed, this section does not apply. |
|---|---|

**WiFi network configuration:**

For the module to connect to your network, create a text file named WIFI.CFG inside the SYS folder of the microSD card, with the network name on one line and the password on the next:

> MyWiFiNetwork
>
> MyPassword

You can add more than one network by repeating the same pattern (name, then password), one after another in the same file:

> HomeWiFi
>
> homepassword
>
> WorkWiFi
>
> workpassword

The module tries the networks in the order they appear, until it connects to the first one available - useful so you do not have to edit the file every time you move to a different location (up to 4 networks).

| | |
|---|---|
| **INFO** | Save the file as plain text, with no formatting. Networks must be 2.4GHz - the WiFi module is not compatible with 5GHz networks. |

**Accessing the file server:**

With the module connected to your network, you need its address to reach it from a browser. There are several ways to find it, from most to least convenient:

**From the ZX81 itself (simplest, no other device needed):**

> LOAD THEN PRINT “*IP”

Displays the module current IP address on screen.

**By name, without typing the IP:**

If your device supports mDNS (built in on macOS, iOS, Android and Linux), go directly to http://sd81booster.local

| | |
|---|---|
| **INFO** | On Windows, if you do not have any Apple software installed, the browser may not resolve “.local” addresses. This is not required for anything else - you can just use LOAD THEN PRINT “*IP” or the direct IP address instead. |

**Alternative: router panel:**

You can also find the assigned IP address in the list of connected devices in your router administration panel (look for “esp32” or similar).

**Available functions:**

- List and browse the SD card content by folder.
- Upload files, including several at once or a whole folder (in desktop browsers).
- Download any file to the device you are browsing from.
- Delete files or folders.
- Create new folders.

**NTP time synchronization:**

The WiFi module can keep the interface’s clock accurate by automatically syncing it with a time server (NTP) as soon as it connects to the network - useful if the RTC battery runs low or simply to avoid setting the date by hand.

To enable it, create a text file called NTP.CFG in the SYS folder of the microSD card:

> SERVER=pool.ntp.org
>
> MODE=SERVER

> UTCOFFSET=1
>
> DST=1

SERVER is the NTP server address. MODE=SERVER enables syncing (use MODE=LOCAL, or don't create the file, to keep it disabled). UTCOFFSET is the time difference from UTC, in hours (can be negative). DST adds one extra hour for summer time - it's a manual switch you flip yourself twice a year, just like any other clock.

You can also edit this configuration, and force an immediate sync, from the module's own web page (the "NTP time settings" link on the file server's front page).

**From BASIC, without needing the web page:**

- LOAD *NTP - forces an immediate sync.
- LOAD *NTP="server" - changes the NTP server.
- LOAD *NTP +n / LOAD *NTP -n - sets the UTC offset, in hours.
- LOAD *SUMMER / LOAD *SUMMER STOP - turns summer time on or off.

| | |
|---|---|
| **INFO** | The inspected firmware attempts synchronization at startup. If it fails, it retries every 60 seconds while connected until one attempt succeeds, then stops automatic retries. If synchronization is unavailable, the RTC keeps its current time. Manual synchronization is also available. |

**Updating the module:** follow section 17 for the separate STM32, FPGA, ROM and WiFi files, their exact locations, and the second startup sometimes required by older modules. The WiFi component version in package 1.7 is 1.4. Initial programming and USB recovery are separate procedures.

## 9.8 Telnet Terminal — TELNET

TELNET is a terminal for connecting to BBSes and telnet servers from the ZX81, through the WiFi module. It shows 80 columns by 24 rows in Superfast mode, with the 256-character CP437 font and ANSI colours, and understands the control sequences BBSes use (ANSI and ADM-3A). It is in Software.zip, in the SD81/TOOLS/TELNET folder: copy the folder to the SD card and load the program:

> LOAD "TELNET.P"

The terminal works like a Hayes modem: AT commands are typed on the terminal screen itself.

| Command | Effect |
|---|---|
| ATDT host:port | Connects to the server (for example ATDT bbs.example.org:23). Answers CONNECT or NO CARRIER. |
| ATH | Hangs up. |
| +++ | Goes back to command mode without hanging up (with one second of silence before and after). |
| ATO | Returns to the connection. |

**Special keys:**

| Key | Action |
|---|---|
| **ENTER + 0** | Exit: hangs up if connected and returns to BASIC. |
| **ENTER + 9** | Local echo on/off, for servers that don't echo what you type. |

| Key | Action |
|---|---|
| **ENTER + 8** | Colours: white on black (at start-up), green on black or yellow on blue. Colours set by the BBS are kept. |
| **SHIFT + 1** | ESC. |
| **SHIFT + ENTER, then a key** | CTRL + key. |
| **ENTER + key** | Symbols: the ones printed on the keys plus @ \ \| ~ ` { } [ ] _ ! # % &. |
| **SHIFT + 0 / SHIFT + 9** | DEL / backspace. |
| **SHIFT + 5/6/7/8** | Left, down, up and right arrows: send the ANSI sequences ESC [ D/B/A/C, which is what BBSes understand. |

The keyboard is the same as in CP/M (see the CP/M keyboard table in Appendix I), except for the arrows. With local echo on, control codes are shown as ^X (^[ is ESC and ^C is CTRL + C). The terminal answers the server's telnet negotiation accepting only ECHO and SUPPRESS-GO-AHEAD, so BBSes don't show odd characters when you connect. It uses memory in blocks 4 and 5 ($8000-$97FF) for the screen, the colours and the font.

## 9.9 Turn-based games over the network

With the WiFi module (9.7), two SD81 Booster can play turn-based games over the internet, each at home and with no server of our own: both interfaces connect as clients to a public mailbox (an MQTT server) and send the moves through it. There are two games, Connect 4 (CONNECT4) and chess (CHESS), and a test chat (NETCHAT). They are in Software.zip, in the folders SD81/TOOLS/CONNECT4, CHESS and NETCHAT. Copy the game's folder to the SD card and load its .P program, which is a small loader: it reads the .BIN with the game from the same directory, so the two files must be together.

You need the WiFi module with the network configured (/SYS/WIFI.CFG, see 9.7) and the FPGA with the Superfast modes. The debugger monitor is not needed.

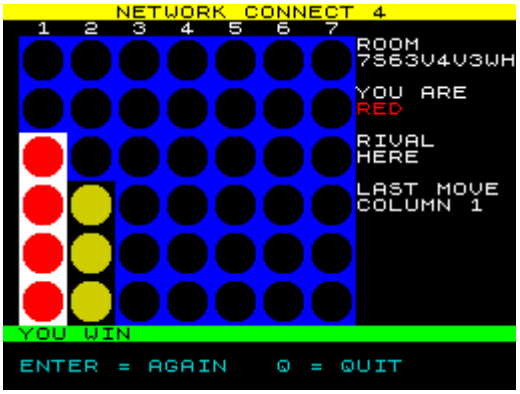
**Starting a game:**

- New game (1): the program creates a room and shows its code, 10 characters long. Tell it to your opponent by any means. Whoever creates the room is player 1: in Connect 4 you start, and in chess you play white.
- Join (2): type the code you were given (SHIFT+0 deletes) and ENTER. You are player 2.
- A room takes two players. A third person typing the code sees THE ROOM IS FULL and cannot enter; a code nobody created gives THERE IS NO SUCH ROOM.
- If you leave the game (0 or Q) or the network drops, you can come back with the same code from «join»: you get your place back and the game carries on where it was. If both dropped, the room is left empty and the program lets you choose which player you return as. If someone drops without saying goodbye, their place looks occupied for a minute or two, until the server notices.
- The two of you do not need to be connected at the same time: if the opponent is away, their last move waits in the mailbox and you get it when you return.
- When the game ends, ENTER offers another one to the opponent.
- Saving and carrying on another day: every time the game changes, the program saves it to a file in the game folder on the SD card (CONNECT4.SAV or CHESS.SAV). If you switch the ZX81 off, when you come back the menu offers «3 = CONTINUE SAVED GAME»: the program goes back into the same room with the same colour, and the game carries on where it was (in chess, with the clocks as they were: the time you spend away does not count). The two players do not need to be connected

at the same time: each one only needs their saved game and the server to keep the other's last move until it is picked up. If the server has forgotten it, whoever comes back first sends their last move again. Starting another game, or joining another, replaces the saved one.

**Connect 4:**

The board is drawn in high resolution: 7 columns by 6 rows in blue, with player 1's counters in red and player 2's in yellow. The numbers 1 to 7 above the columns say which key drops the counter into each one. The counter that has just fallen is highlighted with a white background and, when someone wins, so are the four in a row. The panel on the right shows the room, your colour, whether the opponent is here and the last move. R resends the last move, in case the network failed. Q quits.




Connect 4: the four red counters in the first column win (a capture of the real screen, at its size).

**Chess:**

The game is drawn in high resolution (Superfast HiRes Spectrum): an 8×8 board with its coordinates, the last move highlighted in magenta, the king in check highlighted, and a panel with the room, your colour and the status. Each player sees the board from their own side: black sees it turned round. A move is typed as the origin square and the destination square, for example E2 E4, and ENTER; a key that does not fit in that position is not accepted, and SHIFT+0 deletes the last one. To promote, a fifth letter chooses the piece (Q queen, R rook, B bishop, N knight); without it, queen. 9 resends the last move and 0 quits. All the rules apply except threefold repetition: castling, en passant, promotion, check, checkmate, stalemate, insufficient material and the 50-move rule.




Chess: the starting position (a capture of the real screen, at its size).

**Chess clocks:**

When you create the room you choose each player's time: no clock, 5, 10, 15 or 30 minutes, with no increment. The other player inherits it when typing the code (it goes in its first letter: A no clock, B 5 minutes, C 10, D 15, E 30). Both clocks show in the panel; the clock of the player to move is yellow and red with 30 seconds or less. Whoever runs out of time loses (TIME OUT).

**Test chat (NETCHAT):**

A minimal chat to test the network between two interfaces: one picks new room, the other joins with the code, S sends a line and Q quits.

Note: the moves travel unencrypted through a public MQTT server, signed with a key derived from the room code. That stops a stranger from injecting moves, but it does not hide them: do not send anything sensitive. To write your own game, see Appendix M.

# 10. Sound

The SD81 Booster incorporates an AY-3-8910/12 sound chip emulator, the same used by computers such as the ZX Spectrum 128K or Amstrad CPC. This allows playing music with up to three independent voices, plus noise and envelope effects.

## 10.1 PLAY Command — Music with the AY chip

The PLAY command allows playing music directly from BASIC using a text string that describes the notes, duration, tempo and other parameters. It supports up to three simultaneous voices.

**Basic syntax:**

> LOAD *PLAY “string1”
>
> LOAD *PLAY “string1”,“string2”,“string3”

Each string corresponds to one voice. Between one and three strings can be used simultaneously.

**Simple example — melody in one voice:**

> LOAD *PLAY “T120O45C5E5G9C”

This plays the notes C, E, G, and C (C major chord) at 120 beats per minute in octave 4.

| INFO | For a complete description of all parameters, including envelope effects and noise mode, see Appendix A of this manual. |
|---|---|

## 10.2 VGM Player — Background Music

The SD81 Booster can play VGM (Video Game Music) format files in the background while the ZX81 runs any other program. This allows adding music to your own programs without using any CPU time.

| INFO | Only VGM files containing data from the AY-3-8910 or AY-3-8912 chip are supported. VGMs with other sound chips are not supported. |
|---|---|

**Prepare a VGM file:**

> LOAD *VGM “MUSIC”

Loads the MUSIC.VGM file from the SD and prepares it for playback, without starting it yet.

**Start playback:**

> LOAD *VGM THEN RUN

**Pause and resume:**

> LOAD *VGM THEN PAUSE
>
> LOAD *VGM THEN CONT

**Stop playback:**

> LOAD *VGM THEN STOP

**Activate loop mode** (music restarts when it ends):

> LOAD *VGMLOOP

**Disable loop mode:**

> LOAD *VGMLOOP STOP

| INFO | THEN, RUN, CONT, PAUSE y STOP they are BASIC tokens of the ZX81, they are not written letter by letter. |
|---|---|

**Typical usage example in a BASIC program:**

> 10 LOAD *VGM “MUSIC”
>
> 20 LOAD *VGMLOOP
>
> 30 LOAD *VGM THEN RUN
>
> 40 REM --- program continues while music plays in background ---

## 10.3 PEG Effects Generator — Programmable Sound Effects

The PEG (Programmable Effects Generator) is a small virtual machine integrated in the interface that executes sound effect programs completely independently of the Z80, without consuming any ZX81 CPU time.

The PEG accesses the AY chip registers directly and has up to three parallel execution threads, allowing several effects to play simultaneously.

| INFO | The PEG is primarily aimed at developers. To create PEG programs it is recommended to use the PEG assembler included in the project repository. For a complete description of the instruction set, see Appendix B. |
|---|---|

**Loading a PEG Program in Memory:**

> LOAD *PEG <address>,“<hexadecimal>”

Where <address> is the position in PEG memory (0–255) and <hexadecimal> is the instruction sequence in hexadecimal format. Each instruction occupies 4 hexadecimal characters (2 bytes).

**Start a PEG thread:**

> LOAD *PEG THEN RUN <thread>,<address>

Where <thread> is the thread number (0, 1 or 2) and <address> is the start address of the PEG program.

**Stop, pause, and resume a thread:**

> LOAD *PEG THEN STOP <thread>
>
> LOAD *PEG THEN PAUSE <thread>
>
> LOAD *PEG THEN CONT <thread>

**Loading a PEG program from the SD (LOAD *PEB):**

In addition to inline loading with a hexadecimal string, it is possible to load a compiled PEG program directly from a binary file on the SD:

> LOAD *PEB <address>,“<name>”

Where <address> is the start position in PEG memory (0–255) and <name> is the filename on the SD. The file must have the .PEB extension (PEG binary); if no extension is specified, the interface adds it automatically.

**Example:**

> LOAD *PEB 0,“EFECT”
>
> LOAD *PEG THEN RUN 0,0

This example loads the file EFECT.PEB into PEG memory from position 0 and starts thread 0 from that same address.

| INFO | The .PEB file is the result of assembling a .PEG source with the peg.py assembler, a Python script included in the EXAMPLES/PEG/ folder of the project repository. To compile: python peg.py efect.peg |
|---|---|

## 10.4 Speech Synthesis — SAY Command

The SD81 Booster incorporates a speech synthesiser that can reproduce English phrases directly from BASIC. The synthesiser is based on the phonemes of the SP0256 chip, a speech synthesiser widely used at the time in classic interfaces such as the Currah MicroSpeech for the ZX Spectrum or The Voice for the Videopac G7000/Odyssey 2.

**Syntax:**

> LOAD *SAY “STRING”

The synthesiser analyses the text string and attempts to build the pronunciation by combining English phonemes. The text should be written in English, in uppercase.

**Examples:**

> LOAD *SAY “HELLO WORLD”
>
> LOAD *SAY “ZX81 COMPUTER”
>
> LOAD *SAY “ERROR 5”

Numbers are read in English automatically, from zero to billions.

### Background Playback

By default, the CPU waits for the voice to finish before continuing program execution. If a * is prepended to the start of the string, playback continues in the background and the ZX81 keeps executing BASIC.

> LOAD *SAY “*HELLO WORLD”

| INFO | It is not possible to add custom sounds or phonemes. The available phoneme set is fixed in the microcontroller’s internal memory. |
|---|---|

### How the synthesiser works

The synthesiser decomposes text from left to right, always looking for the longest possible match in its internal dictionary. Recognised words sound better than letter-by-letter phonetic decomposition.

Spaces and punctuation produce pauses of increasing length: a space produces a short pause, a comma a medium pause, and a semicolon or colon a long pause.

| INFO | To improve the pronunciation of unrecognised words, write them phonetically in English. For example, SINCLAIR may sound better as SINCLER. See Appendix C for the complete word dictionary and available phonemes. |
|---|---|

# 11. Memory Management

The SD81 Booster incorporates up to 512 KB of RAM, far above the original 1 KB of the ZX81 or conventional memory expansions. This memory is managed via a memory mapper that divides both the Z80 address space and the interface RAM into 8 KB blocks and pages.

For most BASIC programs it is not necessary to manage memory manually: the interface configures it automatically on boot and the ZX81 BASIC can use the available RAM directly.

## 11.1 Basic Concepts: Blocks and Pages

The Z80 address space (64 KB) is divided into 8 blocks of 8 KB each:

| Block | Address range | Typical use |
|---|---|---|
| **0** | 0000–1FFF | ZX81 ROM (read-only) |
| **1** | 2000–3FFF | Interface extension ROM |
| **2** | 4000–5FFF | Main RAM / screen |
| **3** | 6000–7FFF | Main RAM |
| **4** | 8000–9FFF | Extended RAM |
| **5** | A000–BFFF | Extended RAM |
| **6** | C000–DFFF | Mirror of block 2 (required for video) |
| **7** | E000–FFFF | Mirror of block 3 (required for video) |

The 512 KB of RAM are divided into 64 pages of 8 KB. Each block can point to any of these 64 pages, allowing access to all the memory simply by changing which page is assigned to each block.

| NOTE | Blocks 6 and 7 must be kept as mirrors of blocks 2 and 3 respectively for the ZX81 video system to work correctly. Modifying them without bearing this in mind may cause screen corruption or system instability. |
|---|---|

## 11.2 MAP Command — Assign pages to blocks

To assign a memory page to a given block:

> LOAD *MAP <block>,<page>

Where <block> is a number from 0 to 7 and <page> is a number from 0 to 63.

To read which page is currently assigned to a block:

> LOAD *MAP <block> TO <variable>

**Example — map block 4 to page 10:**

> LOAD *MAP 4,10

From that point on, any read or write to addresses 32768–40959 will access page 10 of the RAM.

**Example — read the page assigned to a block:**

```
10 LOAD *MAP 4 TO A

20 PRINT “Block 4 points to page”;A
```

| INFO | For a complete reference of the paging system, including 512 KB full paging mode and use from machine code, see Appendix D of this manual. |
|---|---|

## 11.3 MC45 Mode — Machine Code in Blocks 4 and 5

By design of the ZX81 hardware, machine code instructions located in blocks 4 and 5 (addresses 32768–49151) are executed incorrectly: opcodes in the ranges 00h–3Fh and 80h–BFh are replaced by NOP, preventing normal code execution in that region.

MC45 mode (Machine Code 4 and 5) deactivates this limitation, allowing any Z80 instruction to be executed in that memory area.

**Activate MC45 mode:**

```
LOAD *MC45
```

**Deactivate it:**

```
LOAD *MC45 STOP
```

| NOTE | MC45 mode is implemented by forcing the Z80’s /HALT pin to zero intermittently and during very brief time intervals, keeping the load on that pin at low levels and making it safe in practice. The interface already includes an internal 680Ω resistor for protection. |
|---|---|

## Extending MC45 to blocks 6 and 7

By default, MC45 only covers blocks 4 and 5 (addresses 32768-49151): in native ZX81 video, the screen is generated by jumping to D_FILE+$8000 and executing NOPs until a real HALT is found (D_FILE itself, in its high mirror), and that address never falls within blocks 4/5 under normal use, so executing real code there is safe.

In the Superfast modes (text, native HiRes or Spectrum HiRes) the video is generated entirely by hardware, without the CPU ever having to “execute” D_FILE — so, with your own code, it could also be safe to execute in blocks 6 and 7 (49152-65535). The SD81 Booster lets you enable this extension with an independent register:

```
POKE 2062, 170 : REM extends MC45 to blocks 6 and 7 as well

POKE 2062, 85 : REM back to the usual restriction (blocks 4 and 5 only)
```

For this to have any effect, LOAD *MC45 must also be active; this register only adds blocks 6 and 7 to the area where MC45 already acts, it does not replace it.

| NOTE | In the default block mapping (without RAM48 enabled), blocks 6 and 7 are a physical mirror of blocks 2 and 3 — i.e. of the memory where the system variables, the BASIC program, the stack and the real D_FILE live. Enabling this extension while blocks 6/7 are still that mirror hangs the system as soon as a Superfast mode is enabled, even with FAST. Only enable it after remapping blocks 6 and/or 7 (with MAP) to free SRAM pages dedicated exclusively to your own code. |
|---|---|

## 11.4 Boot with Alternative ROM

The SD81 Booster allows booting with a different ROM than the standard ZX81 ROM without needing to use any command. Simply place one or more ROM files in the /SYS/ folder on the SD card with names /SYS/0.ROM through /SYS/9.ROM.

**To boot with an alternative ROM:**

1. Hold down the number corresponding to the desired ROM file while powering on the ZX81.
2. Release the key once the computer has booted.

| INFO | **This feature is very useful for testing alternative or modified ROMs without reprogramming any chip. The standard ZX81 ROM always loads if no key is pressed during boot.** |
|---|---|

## 11.5 User-Definable Characters (128C / 64C / 256C)

By default, the ZX81 has 64 characters defined by the ROM. The SD81 Booster allows you to expand this set to 128 characters, all of which are completely redefinable, by writing to the memory area between addresses 15360 and 16383 (3C00h–3FFFh).

**Activate 128-character mode:**

> LOAD *128C

**Return to standard 64-character mode:**

> LOAD *64C

**Activate 256-character mode:**

> LOAD *256C

This mode only works in Superfast text mode. It extends the set to 256 characters, all redefinable, in the memory area between addresses 14336 and 16383 (3800h–3FFFh) — twice the space of 128-character mode, aligned to 2K instead of 1K. As with 128C, the whole block is preloaded with the ROM character set at boot, so enabling the mode without having redefined anything does not change what is shown on screen. Activating “128C” or “64C” disables “256C”.

With 256-character mode active, the Chroma mode 0 colour table (indexed by character code) also grows from 1K to 2K (14336–14463, $C000-$C7FF), so each of the 256 codes gets its own colour entry. Mode 1 (attribute file) is unaffected -- it is indexed by screen position and already works the same with 128 or 256 characters.

| INFO | **In 128-character mode, the upper 64 characters are automatically shown in inverse video by the hardware. To display them in normal video you must store the inverted graphic in the corresponding position.** |
|---|---|

If you need to restore the original character set after modifying it:

> LOAD *LDIR 7680,15360,512
>
> LOAD *LDIR 7680,15872,512

| INFO | **Activating 128-character mode is incompatible with the HRG internal character generator (see section 10.6). Both modes cannot be active simultaneously.** |
|---|---|

### Example: Spectrum-style start screen

The following program illustrates the use of 128-character mode to load an alternative character set from the SD and display a screen in ZX Spectrum style:

```
5 LOAD *128C

20 LOAD FAST “SPEC-81-128.BIN” CODE 15360

30 CLS

35 POKE 16418,0

40 PRINT AT 23,1;CHR$ 8;” 1982 S[INCLAIR] R[ESEARCH] L[TD].”

45 POKE 16418,2

50 IF INKEY$=“” THEN GOTO 50
```

| INFO | **Texts in brackets indicate inverse video: S[INCLAIR] means the S is normal and INCLAIR is in inverse video; R[ESEARCH] means R is normal and ESEARCH is in inverse video; L[TD] means L is normal and TD is in inverse video. To enter inverse video characters press SHIFT + 9 before each character. CHR$ 8 corresponds to the ZX81 grid graphic character (code 8, the © symbol in the Spectrum character set).** |
|---|---|

**Line by line explanation:**

- Line 5: activates 128-character mode.
- Line 20: loads the file SPEC-81-128.BIN from the SD at address 15360 (3C00h), the user-definable character area. This file contains the ZX Spectrum character set.
- Line 30: clears the screen.
- Line 35: activates Superfast mode (POKE 16418,0) to free the CPU from video control.
- Line 40: prints on line 23 the Spectrum copyright message, using CHR$ 8 as the © symbol and the signature words in inverse video.
- Line 45: restores standard video mode (POKE 16418,2).
- Line 50: waits indefinitely for any key press.

# 12. Advanced BASIC Extensions

This section covers additional BASIC extension commands of the SD81 Booster oriented primarily towards programming: string manipulation, memory access, input/output port communication, and directory access from a program.

## 12.1 String Manipulation

**Invert characters in a string (*INV):**

Inverts bit 7 of all characters in a string variable, turning normal characters into inverse and vice versa:

```
LOAD *INV A$
```

String slices are also supported:

```
LOAD *INV A$(2 TO 7)
```

**Force inverse video on a string (*BOLD):**

Forces bit 7 of all characters in a string variable to 1, putting all characters in inverse video:

```
LOAD *BOLD A$
```

| INFO | To put characters in normal video from an inverse string, first apply *BOLD and then *INV to invert the result. |
|---|---|

## 12.2 Memory Block Copy and Fill

**Copy a memory block in ascending order (*LDIR):**

```
LOAD *LDIR <source>,<dest>,<length>
```

Copies <count> bytes from <source> to <dest> in ascending order. Equivalent to the Z80 LDIR instruction.

**Copy in descending order (*LDDR):**

```
LOAD *LDDR <source>,<dest>,<length>
```

Same as *LDIR but in descending order. Equivalent to the Z80 LDDR instruction.

| INFO | When source and destination overlap, the copy order matters: use *LDIR if the destination is before the source in memory, and *LDDR if it is after, to avoid data being overwritten during the copy. |
|---|---|

**Loading Hexadecimal Data into Memory (*HEX):**

```
LOAD *HEX <address>,“<hexadecimal>”
```

For example, LOAD *HEX 30000,“0A014020” loads the values 10 (0Ah), 1, 64 (40h) and 32 (20h) starting at address 30000.

## 12.3 Machine Code Execution

**Execute a machine code routine (LOAD USR):**

```
LOAD USR <address>
```

Equivalent to RAND USR but with three important advantages:

- Does not modify the random number generator.
- The routine is called from the top level of BASIC, leaving the alternate registers BC’, DE’ and HL’ available and the stacks clean.
- The rest of the line is not parsed syntactically, allowing the routine to perform its own parameter parsing via RST 18h and RST 20h.

Upon entry to the routine, the BC register contains the address called.

| INFO | Limitation with numeric literals: if the routine uses the ROM’s SCANNING routine to parse parameters, expressions with direct numeric literals (e.g. USR 40000) will not work correctly. Use VAL “number” or CODE “character” as a workaround. |
|---|---|

## 12.4 Input/Output Port Access

**Write to a port (*OUT):**

> LOAD *OUT <port>,<value>

**Read from a port (*IN):**

> LOAD *IN <port> TO <variable>

| INFO | TO is the BASIC token (SHIFT + 4), not typed letter by letter. |
|---|---|

## 12.5 16-bit Memory Access

**Read a 16-bit memory value (LOAD PEEK):**

> LOAD PEEK <address> TO <variable>

Reads two consecutive bytes from memory starting at <address> and stores them as a 16-bit value in <variable>.

**Writing a 16-bit value to memory (LOAD THEN POKE):**

> LOAD THEN POKE <address>,<value16>

**Set the upper memory limit for BASIC (LOAD THEN CLEAR):**

> LOAD THEN CLEAR <address>

Sets the last RAM address available for BASIC (RAMTOP ends up at that address plus one: LOAD THEN CLEAR 32767 restores the standard value for a 16K ZX81, RAMTOP = 32768). Unlike the standard CLEAR command, this does not clear variables; it only clears the GOSUB stack. Use a separate CLEAR if you also want to clear variables.

## 12.6 Directory Access from a Program

These commands allow reading the content of an SD directory from within a BASIC program, useful for building file selection menus.

**Open a directory for reading (*OPENDIR):**

> LOAD *OPENDIR <string>

The string can include a full path and wildcards. If wildcards are used, the maximum number of retrievable entries is 512. This command leaves the directory ready to use with *ROW.

**Read a directory entry (*ROW):**

> LOAD *ROW <number> TO <variable$>

Reads the directory entry with the given number (starting from 1) and stores it in the string variable. If the number is out of range, returns an empty string.

**Example — file selection menu in BASIC:**

```
10 LOAD *OPENDIR “JUEGOS/*.P”
20 FOR I=1 TO 10
30 LOAD *ROW I TO A$
40 IF A$=“” THEN GOTO 70
50 PRINT I;“.”;A$
60 NEXT I
```

```
70 INPUT “Select:”;N
80 LOAD *ROW N TO A$
90 LOAD FAST A$
```

# 13. Program Examples

This section contains example programs that illustrate the use of the SD81 Booster’s main features. All are written in standard ZX81 BASIC with the interface extensions.

| INFO | Texts in brackets (e.g. [SINCLAIR]) are entered in inverse video on the ZX81 by pressing SHIFT + 9 before each character. CHR$ 8 corresponds to the ZX81 grid graphic character (code 8). |
|---|---|

## 13.1 Real Time Clock (RTC)

Demonstrates three clock adjustment formats: full date and time, date only and time only.

```
10 LET A$=“2025-11-10 13:48:00.00”
15 LOAD *RTC
16 PRINT
20 LOAD *RTC=A$
25 LOAD *RTC
26 PRINT
30 LOAD *RTC=“2026-11-10”
35 LOAD *RTC
36 PRINT
40 LOAD *RTC=“12:20”
45 LOAD *RTC
46 PRINT
50 LOAD *RTC=“13:20:35”
55 LOAD *RTC
56 PRINT
```

Displays the current time (line 15), then adjusts it via a string variable (line 20), then changes only the date (line 30), then only the time in short format (line 40) and finally with hours, minutes and seconds (line 50). After each adjustment it prints the result to verify it.

## 13.2 RTC Battery Status (BAT)

```
10 LOAD *BAT TO A$
20 PRINT A$
```

Reads the real time clock battery status and displays it on screen. If the battery is low, the displayed value will indicate it.

## 13.3 MC45 Mode Check

Checks whether machine code mode in blocks 4 and 5 is active by loading a small assembly routine and executing it:

```
10 LOAD *HEX 40000,“010203C9”

20 IF USR 40000=770 THEN GOTO 100

30 PRINT “MC45 INACTIVE”

40 STOP

100 PRINT “MC45 ACTIVE”
```

Loads a Z80 routine at address 40000 (blocks 4-5) that returns the value of BC on exit (C9=RET). If MC45 is not active, the instructions in that area will not execute correctly and the value returned will not be 770. If MC45 is active, the routine executes correctly and jumps to line 100.

## 13.4 Superfast Mode — Speed Demonstration

Visually compares the refresh speed in standard ZX81 SLOW mode versus the SD81 Booster Superfast mode:

```
4 SLOW

5 PRINT “ZX81 SLOW MODE”

6 PAUSE 250

7 POKE 2045,85

8 POKE 16418,0

9 CLS

10 GOSUB 1000

15 CLS

20 POKE 2045,170

30 PRINT “SD81-BOOSTER SUPER FAST MODE”

31 PAUSE 250

35 CLS

36 FAST

37 GOSUB 1000

40 GOTO 40

75 POKE 1024,2

76 POKE 1024,3

77 POKE 1024,4

1000 PRINT “[CHR$ 0]123456789ABCDEFGHIJKLMNOPQRSTUV”;

1010 FOR N=1 TO 22

1020 PRINT “0123456789ABCDEFGHIJKLMNOPQRSTUV”;
```

```
1030 NEXT N
1040 PRINT “0123456789ABCDEFGHIJKLMNOQRSTUV[CHR$ 0]”;
1050 RETURN
2000 PRINT “[CHR$ 0]123456789ABCDEFGHIJKLMNOPQRSTUV”;
2010 FOR N=1 TO 22
2020 PRINT “0123456789ABCDEFGHIJKLMNOPQRSTUV”;
2030 NEXT N
2040 PRINT “0123456789ABCDEFGHIJKLMNOQRSTUV[CHR$ 0]”;
2050 RETURN
```

| INFO | **[CHR$ 0] in lines 1000, 1040, 2000 and 2040 represents character 0 in inverse video as seen in the original listing.** |
|---|---|

The program first shows a full screen of text in standard ZX81 SLOW mode, where the CPU dedicates time to video refresh. Then it activates Superfast mode (POKE 2045,170) and repeats the same screen fill with FAST, showing the speed difference.

## 13.5 Spectrum Image Loading

Loads a Spectrum format image (.SCR) from the /SCR/ folder on the SD and displays it using the interface’s HiRes Spectrum mode:

```
5 LOAD *CD “/SCR”
15 LET HFILE=32768
20 POKE 2044,HFILE/256
25 POKE 2045,172
30 LOAD *OUT 32751,39
40 LOAD *OUT 251,6
50 LOAD FAST “Z.SCR” CODE HFILE
190 IF INKEY$=“” THEN GOTO 15
200 POKE 2045,85
210 LOAD *OUT 32751,0
```

**Line by line explanation:**

- Line 5: changes to the /SCR directory on the SD.
- Line 15: defines HFILE=32768 (8000h), start of block 4.
- Line 20: writes the high byte of the screen file address to the interface register.
- Line 25: activates Superfast HiRes Spectrum mode (POKE 2045,172).
- Lines 30-40: configures border colour registers via the interface output port.
- Line 50: loads the file Z.SCR at address HFILE (32768).
- Line 190: waits for any key press; on press returns to line 15 to reload.

- Line 200: deactivates Superfast mode.
- Line 210: restores the output register to 0.

## 13.6 Wide Screen Modes — COL80 / COL70 / COL32 Commands

The SD81 Booster lets you widen the text screen in Superfast mode, from the ZX81's standard 32 columns to 70 or 80. Unlike a direct POKE 2045, these commands also adapt the ROM's own routines (PRINT, PRINT AT, CLS, SCROLL, the listing editor) to the new width, so standard BASIC works exactly as before, just with more columns per line.

**Activate 80 columns (7-pixel characters):**

```
LOAD *COL80
```

**Activate 70 columns (8-pixel characters):**

```
LOAD *COL70
```

**Return to 32 columns, native video:**

```
LOAD *COL32
```

The 70- and 80-column modes occupy exactly the same physical screen width (80 characters at 7 pixels equal 70 at 8), so switching between them does not move or resize the picture: only the number of text columns that fit changes. All three commands switch to FAST mode automatically (in Superfast mode the interface generates the video, not the CPU, so SLOW gains nothing); COL32 does the opposite and returns to SLOW.

Switching modes clears the screen (equivalent to CLS): the previous mode's screen content is not preserved, because the width of each row in the screen area changes.

**TAB and PLOT:**

TAB and PLOT work in all three modes, like the rest of the printing commands (PRINT, PRINT AT, CLS, SCROLL). In 70 and 80 columns, a TAB beyond the line width gives error B instead of wrapping round as in 32 columns. PLOT and UNPLOT accept X from 0 to 139 in 70 columns and from 0 to 159 in 80.

# 14. Error Codes

When an error occurs, the ZX81 displays a code at the bottom of the screen followed by the line number where it occurred. The error codes related to the SD81 Booster are:

| Code | Meaning |
|---|---|
| **A** | Invalid argument (incorrect filename, parameter out of range, or hexadecimal string with incorrect length in *PEG or *HEX). |
| **D** | The user pressed BREAK to interrupt an operation (e.g. during a directory listing). |
| **G** | File not found on the SD card. |
| **H** | Error accessing the SD card. Check it is correctly inserted and formatted in FAT32. |
| **I** | I/O error on the SD card during a read or write operation. |
| **J** | Disk full: not enough space on the SD card to save the file. |
| **K** | File or directory already exists with that name. |
| **L** | File name that is too long or with characters that are not allowed. |

| Code | Meaning |
|---|---|
| **M** | The directory is not empty (when trying to delete with *RD). |
| **N** | Permission denied or write-protected file. |
| **P** | Could not load: a .B81 file with errors (details with LOAD THEN PRINT “*B81ERR”), or a .Z81 snapshot without the debugger (/SYS/DEBUG.BIN is missing), with the debugger stopped, or with a snapshot in progress. |

| INFO | **If you get error H repeatedly, remove the SD card, check it is formatted as FAT32, and reinsert it. If the error persists, try a different card.** |
|---|---|

# 15. For Programmers

This section is aimed at developers who wish to take advantage of the SD81 Booster’s advanced capabilities from Z80 machine code, including the expansion ROM routines and the MCU communication system.

## 15.1 Expansion ROM Map

The expansion ROM occupies block 1, starting at address 8192 (2000h):

| Address | Content / Routine |
|---|---|
| 2000h | String ‘SD81’ in ZX81 character encoding |
| 2004h | ROM version byte (high nibble = major, low nibble = minor). PEEK 8196. |
| 2005h | Routine: returns in HL the return address of the caller |
| 2006h | Routine: Runs JP (HL) — emulates CALL (HL) |
| 2007h | GetMCUVersion — MCU version in BC (B=0, C=version). USR 8199. |
| 200Ah | WaitClkDiff — wait for a different clock bit than C bit 7 |
| 200Dh | WaitClkEq — wait clock bit equal to C bit 7 |
| 2010h | OutWaitDiff — sends A to the data port and waits for different clock |
| 2013h | OutWaitEq — sends A to the data port and waits for equal clock |
| 2016h | WaitDiffBrk — like WaitClkDiff but allows BREAK (does not return if pressed) |
| 2019h | WaitEqBrk — like WaitDiffBrk but expect equality |
| 201Ch | SendString — sends string to MCU prefixed by length. B=length, DE=address |
| 201Fh | SendStrLoop — sends B bytes to MCU from DE without waiting for clock changes |
| 2022h | ReportStatus — reads a byte from the MCU; if ≠0, raises BASIC error (1=G, 2=H, ...) |
| 2025h | PrintBPaged — prints character with paging. B=character (0–63 or 128–191) |
| 2028h | Cmd64C — Activate 64 character mode (= LOAD *64C) |
| 202Bh | Cmd128C — Activate 128 character mode (= LOAD *128C) |
| 202Eh | GetPhase — clock status in C bit 7 bit |
| 2031h | GetData — reads the MCU’s data port. Result in A. Does not modify flags. |

| Address | Content / Routine |
|---|---|
| 2034h | SD81_RESET — RESET entry point. Requires NMI and DI disabled; HL=continuation address. |
| 2037h | SD81LOADCMD — entry point for the extended LOAD command |
| 203Ah | SD81SAVECMD — entry point for extended SAVE command |
| 203Dh | SD81RUNCMD — entry point for extended RUN command |

**Get version from BASIC:**

```
10 LET V=USR 8199
20 LET MAJ=INT(V/16)
30 LET MIN=V-16*MAJ
40 PRINT “VERSION MCU:”;CHR$(MAJ+28);“.”;CHR$(MIN+28)
```

## 15.2 I/O Ports and MCU Protocol

| Port | Function |
|---|---|
| **E7h** | Memory Mapper |
| **A7h** | MCU data port (read and write). |
| **AFh** | MCU control port (write=reset MCU; bit 7 on read=clock bit, bits 6..1 are the VSYNC counter since the last read and bit 0 on read indicates the instantaneous state of VSYNC) |

### VSYNC Synchronisation

Bit 0 of port AFh reflects the status of the vertical sync interrupt (VSYNC). This allows the CPU to wait for the start of screen refresh precisely, without needing interrupts or the HALT command.

Bits 6..1 are the VSYNC counter since the last read and bit 0 on read indicates the instantaneous state of VSYNC)

On the ZX Spectrum, many games used HALT or an IM1/IM2 interrupt routine to synchronise with the vertical scan. On the SD81 Booster this mechanism replaces that functionality and is especially useful when porting Spectrum games.

**Example VSYNC wait loop in Z80 assembly:**

```
WAIT_VSYNC:
in a,(0AFh) ; Read MCU Data Port
and 01h ; isolate bit 0 (VSYNC)
jr nz,WAIT_VSYNC ; wait until VSYNC = 0
WAIT_VSYNC2:
in a,(0AFh)
and 01h
jr z,WAIT_VSYNC2 ; wait edge (VSYNC = 1)
; Synchronized with the start of the frame
```

| NOTE | Writing any value to port AFh causes a software reset of the MCU. This should not be done while the MCU is saving or copying a file. |
|---|---|

The communication is synchronised via the clock bit (bit 7 of port AFh), which is toggled with each read or write on A7h. The Z80 must wait for the bit to change before the next operation.

**Example — GETBYTE command (index 16 of MCU internal memory):**

```
in a,(0AFh) ; Read Start Clock Bit

ld c,a

ld a,20h ; GETBYTE code

out (0A7h),a ; Send command

WAIT1: in a,(0AFh)

xor c

jp p,WAIT1 ; Wait for a different clock

ld a,16 ; index

out (0A7h),a ; send parameter

WAIT2: in a,(0AFh)

xor c

jp m,WAIT2 ; wait for equal clock

in a,(0A7h) ; read response — result in A
```

## 15.3 Memory Mapper (port E7h)

In simple paging mode (up to 256 KB), the 8 bits written to port E7h are interpreted as follows:

| Bits | Function |
|---|---|
| **D2, D1, D0** | Block number (0–7) |
| **D7, D6, D5, D4, D3** | Page number (0–31) |

To access all 64 pages of full mode (512 KB), use the OUT (C),r instruction with the page number in B and the block number in another register. If A contains the block number (0–7) and B the page (0–63):

```
ld c,0E7h ; mapper port

out (c),a ; select page B in block A
```

The change between simple and full mode is done via an MCU command (FULLPAGING and HALFPAGING commands, see section 15.5).

## 15.4 Debug Console (USB-C Port)

The USB-C port on the interface also works as a serial debug port. When connected to a computer, the operating system detects a virtual serial port associated with the CH340G chip. On some systems the CH340 drivers may need to be installed.

**Connection parameters:**

| Parameter | Value |
|---|---|
| **Speed** | 115200 baud |
| **Data bits** | 8 |
| **Parity** | None |
| **Stop bits** | 1 |
| **Flow control** | None |

With any serial terminal program (PuTTY on Windows, minicom on Linux, CoolTerm on macOS, or the Arduino IDE's own Serial Monitor) it is possible to monitor in real time the MCU messages, including: boot progress, SD access errors, firmware update progress, and debug messages from the filesystem, VGM, PEG and speech synthesis.

The console also accepts the hardware debugger's commands (see Appendix L); h shows their help.

| INFO | **The production firmware emits basic status messages via the serial port. Recompiling the firmware with the DEBUG macro active produces much more detailed output, useful for advanced diagnostics and development.** |
|---|---|

## 15.5 Complete MCU Command Table

Commands are sent to the MCU by writing their code to data port A7h, following the clock bit synchronisation protocol described in section 15.2. All string parameters are preceded by a byte with the length.

### Error codes returned by commands

| Code | Meaning |
|---|---|
| **0** | Success |
| **1** | File or directory not found |
| **2** | Not a directory |
| **3** | Operation error (could not create/delete/rename) |
| **4** | File or directory already exists |
| **5** | File too large |
| **6** | Could not create destination file |
| **7** | Write error |
| **8** | Partial read error |
| **12** | No VGM file open |
| **13** | Operation not permitted in T81 directory |
| **14** | Invalid joystick parameter |

## System commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **0** | **NOP** | **—** | **—** | **No operation. Only synchronises the clock.** |
| 1 | VERSION | — | 1 byte: version | Returns the MCU version. Same format as byte at 2004h. |
| 32 | GETBYTE | 1 byte: index (0–255) | 1 byte: value | Reads a byte from MCU internal memory. Indices 0–127: volatile system variables. Indices 128–255: EEPROM (persistent). |
| 33 | SETBYTE | 1 byte: index + 1 byte: value | — | Writes a byte to internal memory. |

## Filesystem commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **2** | **PWD** | **—** | **String + EOT + status** | **Returns the current directory in ZX81 encoding.** |
| 3 | CD | String: path | Status | Changes the current directory. Accepts absolute (/) and relative paths. |
| 4 | DEL | String: filename | Status | Deletes a file from the current directory. No wildcards. |
| 5 | MKDIR | String: filename | Status | Creates a subdirectory. |
| 6 | RMDIR | String: filename | Status | Removes an empty directory. |
| 7 | MOVE | String: source + String: destination | Status | Renames or moves a file. |
| 8 | COPY | String: source + String: destination | Status | Copies a file. Date/time is not preserved. |
| 9 | LOAD | String: filename | 2B length + N bytes + Status | Loads a file. .P/.81: calculates real size. .ROM: loads at address 0 and resets. .WAV: plays audio. |
| 10 | SAVE | String: filename + 2B length + N bytes | Status | Saves a data block as a file on the SD. |
| 11 | TYPE | String: filename | String char by char + EOT + Status | Sends the content of a text file. With * searches in /MAN/ with .TXT extension. |
| 12 | DIR | String: path/wildcard | String char by char + EOT + Status | Lists the directory, including file sizes. |
| 14 | FREE_TXT | — | String + EOT + Status | Returns SD total and free space as text. |

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| 15 | FREE | — | 4B total + 4B free + Status | Space total and free in KB as 32-bit little-endian values. |
| 16 | OPENDIR | String: path/wildcard | Status | Opens a directory and builds an internal array (max. 512 entries). |
| 17 | GETROWLEN | 2B: index | 1B: length + Status | Length of the name of entry index in the array opened with OPENDIR. |
| 18 | GETROW | 2B: index | 1B: length + N bytes + Status | Name of entry index in ZX81 encoding. Index 0 = current directory. Directories between < and >. |
| 53 | F_OPEN | Handle(0..3)+name in ASCII | 1B: status | Open a big file (32 bits size) in a specified file handle (0..3). Name is a ASCII Pascal string. |
| 58 | F_OPEN_ZX81 | Handle(0..3)+name in ZX81 | 1B: status | Open a big file (32 bits size) in a specified file handle (0..3). Name is a ZX81 Pascal string. |
| 54 | F_SEEK | Handle(0..3)+Offset (4 bytes Little endian) | 1B: status | Move read/write pointer to the specified position by offset |
| 55 | F_READ | Handle(0..3)+Count(2B Little Endian) | count bytes + 1B:status | Read count bytes. It Will send count bytes allways padding with zeroes if needed |
| 56 | F_WRITE | Handle(0..3)+Count(2B Little Endian)+info to write (count bytes) | 1B:status | Write count bytes. |
| 57 | F_CLOSE | Handle(0..3) | 1B:status | Close file |
| 59 | F_STAT | Handle(0..3) | 4B:size+2B:date+2B:time+1B:status | Returns the size (32 bits) and creation date/time (FAT format) of a file already opened with F_OPEN. |

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| 75 | LOAD_Z81 | String: name | Status (1B) + IM (1B) | LOAD *Z81 (see 7.7), through the debugger monitor (Appendix L). The MCU reads the whole file before answering: Status 0 = the ROM fires the trap and the MCU loads everything when it stops; 1 no such file, 2 no [MEMORY], 3 malformed file, 10 not possible now (no /SYS/DEBUG.BIN, debugger stopped or snapshot in progress). IM is the snapshot's interrupt mode. Up to 1.6 command 70 was a loader of its own, without the monitor; it has been removed. |

## Hardware control commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **19** | **ENABLE_MC45** | **—** | **—** | **Activates MC45 mode (machine code in blocks 4 and 5).** |
| 20 | DISABLE_MC45 | — | — | Deactivates MC45 mode. |
| 21 | JOY | String: 5 bytes of ZX81 keys | Status | Configures the joystick mapping: left, right, up, down, fire. |
| 27 | SEL_128CHARS | — | — | Activates 128-character mode. Equivalent to LOAD *128C. |
| 28 | SEL_64CHARS | — | — | Activates standard 64-character mode. Equivalent to LOAD *64C. |
| 29 | FULLPAGING | — | — | Activates full paging mode (512 KB, 64 pages). |
| 30 | HALFPAGING | — | — | Activates simple paging mode (256 KB, 32 pages). |
| 48 | ENABLE_48K | — | — | Activates 48 KB extended RAM mode. Equivalent to LOAD *RAM48.. |
| 49 | DISABLE_48K | — | — | Deactivates extended RAM mode. Equivalent to LOAD *RAM48 STOP. |
| 65 | SEL_256CHARS | — | — | Activates 256-character mode (Superfast text only). Equivalent to LOAD *256C. |
| 68 | ROMLOCK_ON | — | — | Locks the block 0 configuration ports (2041-2062, 2090-2098, sprites): writes are ignored. Equivalent to LOAD *ROMLOCK. |
| 69 | ROMLOCK_OFF | — | — | Enables the block 0 configuration ports again. Equivalent to LOAD *ROMLOCK STOP. |

## Speech synthesis commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **22** | **BINARY_SAY** | **String: allophone bytes** | **Status** | **Plays allophones in binary format. Synchronous (blocks until finished).** |
| 23 | SAY | String: ASCII text | Status | Converts text to phonemes and plays it. With * as first character: background. Equivalent to LOAD *SAY. |

## AY / sound commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **24** | **AY_SET_REG** | **1B: register (0-15) + 1B: value** | **—** | **Writes a value to an AY emulator register.** |
| 25 | AY_GET_REG | 1B: register (0-15) | 1B: value | Reads the current value of an AY emulator register. |
| 26 | AY_PLAY | String: channel A + String: B + String: C | Status | Plays up to three simultaneous PLAY strings. With * in ch A: background. Equivalent to LOAD *PLAY. |

## VGM commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **34** | **PLAY_VGM** | **String: filename** | **Status** | **Opens and starts playing a VGM file in background. Adds .vgm if no extension.** |
| 35 | STOP_VGM | — | — | Stops VGM playback and resets the AY emulator. |
| 36 | PAUSE_VGM | — | — | Pauses VGM playback. |
| 37 | CONT_VGM | — | — | Resumes paused VGM playback. |
| 38 | LOOP_VGM | 1B: mode (0=no loop, 1=loop) | — | Sets the VGM player loop mode. |

## PEG commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **40** | **LOAD_PEG** | **1B: address + String: hex data** | **—** | **Loads PEG instructions into generator memory. 2 bytes per instruction in little-endian.** |
| 41 | PLAY_PEG | 1B: thread (0–2) + 1B: address | — | Starts execution of a PEG program on the indicated thread. |
| 42 | STOP_PEG | 1B: thread (0-2) | — | Stops and resets the indicated PEG thread. |
| 43 | PAUSE_PEG | 1B: thread (0-2) | — | Pauses the indicated PEG thread. |
| 44 | CONT_PEG | 1B: thread (0-2) | — | Resumes the indicated PEG thread. |
| 45 | SDLOAD_PEG | String: name + 1B: address | Status | Loads a .PEB file from SD into PEG memory. Maximum size: 512 bytes. |

### RTC and battery commands

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **50** | **RTC** | **String: date/time (or empty to read)** | **If read: ZX81 String + Status. If write: Status** | **Without params: returns date/time. With params: sets the clock. Formats: YYYY-MM-DD HH:MM:SS.CC / YYYY-MM-DD HH:MM:SS / YYYY-MM-DD / HH:MM:SS.CC / HH:MM:SS / HH:MM.** |
| 52 | BAT | — | 5 bytes ASCII + Status | Returns RTC battery level as a 5-character string in format V.mmm (ZX81 encoding). |

### Network commands (NET)

| Code | Name | Parameters | Response | Description |
|---|---|---|---|---|
| **66** | **NET_READ** | **1B: max (0–255)** | **count (1B) + data[count] + avail (1B) + status (1B)** | **Reads up to max bytes received on the socket. count can be 0 even with a live connection — data arrives in bursts. avail = bytes still pending after this read, saturated at 255. With max=0 this becomes a cheap “anything there?” check that only refreshes status, without transferring data.** |
| **67** | **NET_WRITE** | **1B: count + data[count]** | **accepted (1B) + status (1B)** | **Sends count bytes to the socket. accepted may be less than count if the output buffer is full — the Z80 has to resend the rest.** |

The status field is the same for both commands:

| Value | Meaning |
|---|---|
| **0** | not connected |
| **1** | connecting |
| **2** | connected |
| **3** | error / the connection dropped |

# 16. Troubleshooting

## 16.1 Interface Does Not Boot or ZX81 Freezes

| Symptom | Solution |
|---|---|
| **ZX81 shows nothing on power on** | Check the interface is correctly inserted in the expansion port. Disconnect and reconnect with the ZX81 powered off. |
| **STAT LED does not light up** | Check the SD card is inserted and contains the SYS folder with its complete contents. Without it the interface will not boot. |
| **STAT LED blinks Blue/Red during boot** | Error initialising the SD card. Check it is correctly inserted and formatted in FAT32. |
| **STAT LED blinks Orange/Red during boot** | Error writing ROM to RAM. Check the SYS folder contains the necessary ROM files. |
| **STAT LED blinks Orange during boot** | MCU is waiting for FPGA response. If the blink does not stop, it may indicate a hardware problem. |

| Symptom | Solution |
|---|---|
| **Blank screen or noise** | Make sure the expansion connector pins are not bent or dirty. |
| **ZX81 boots but interface commands do not work** | Check the firmware version with LOAD *VER. To update, copy firmware.bin to the SD root and power on the ZX81. |

## 16.2 microSD Card Problems

| Symptom | Solution |
|---|---|
| **Error H when trying to load** | Check the SD is correctly inserted. Remove and reinsert it. If the error persists, reformat the card in FAT32. If it continues failing, the SD card may be defective. |
| **SD not recognised** | Check it is formatted in FAT32 (not exFAT or NTFS). |
| **Error G when loading a program** | The file does not exist with that name. Use LOAD *DIR to see exact names. |
| **Error J when saving** | The SD card is full. Use LOAD *FREE to check available space. |

## 16.3 Clock Loses Time on Power Off

The real time clock is powered by the CR2032 button battery. If the clock systematically loses time when the ZX81 is powered off, the battery probably needs replacing.

Check the charge level with:

> LOAD *BAT

If the voltage is below approximately 2.5V, replace the battery with a new CR2032. The battery is located on the interface board and can be removed with a thin flat tool.

## 16.4 Joystick Does Not Respond

| Symptom | Solution |
|---|---|
| **Joystick does nothing** | Configure the mapping with LOAD *JOY “QAOP” (up/down/left/right/fire) or another mapping depending on the game, before launching the program. |
| **Only some directions work** | Check the configuration string has exactly 5 characters. |
| **Joystick moves but does not fire** | Check the fifth character of the JOY string corresponds to the game’s fire key. |

## 16.5 Sound Does Not Work

| Symptom | Solution |
|---|---|
| **No sound with LOAD *PLAY** | Check the television or monitor is connected to the interface’s SCART connector and that the SCART audio channel is not muted. |
| **Speech is not intelligible** | Try short phrases in English. Spell words phonetically if the result is not satisfactory. |
| **VGM does not play** | Check the file is a VGM with AY chip data only. VGMs with other chips are not compatible. |

## 16.6 Firmware Update Errors

| Symptom | Solution |
|---|---|
| **STAT LED blinks Blue/Red on boot with firmware.bin on SD** | Error initialising SD card before update. Remove the SD, check FAT32 format and try again. |
| **STAT LED blinks White/Red after trying to update** | Error during update. The firmware.bin file remains on the SD. Check the file is not corrupted and power on the ZX81 again to retry. |
| **STAT LED stays solid yellow indefinitely** | The meaning depends on the phase: it may indicate MCU writing or a WiFi error. See section 17.6 and do not interrupt an active write. |
| **After update the interface does not respond** | Check with LOAD *VER that the version is correct. If the interface does not boot, follow the emergency USB recovery procedure described in section 17. |

## 16.7 Using the Debug Console as a Diagnostic Tool

When the STAT LED shows an error but the cause is not clear, the USB-C debug console can provide very valuable additional information.

**How to connect:**

1. Connect a USB-C cable between the interface and the computer.
2. Open a serial terminal program (PuTTY, Tera Term, minicom, the Arduino IDE’s Serial Monitor...) and connect to the CH340 COM/serial port with the parameters: 115200 baud, 8N1, no flow control.
3. Power on the ZX81 with the interface connected.
4. Observe the messages that appear in the terminal during boot and normal operation.

The production firmware emits basic status messages that allow identifying which boot phase fails, whether the SD card is recognised correctly, the result of firmware updates and other relevant events.

| | |
|---|---|
| **INFO** | **For developers: recompiling the firmware with the DEBUG macro active produces much more detailed output, including flash operation progress, AY register status, and the details of each command received from the Z80.** |

## 16.8 Hardware Test Program — SD81TEST

SD81TEST checks the interface systematically: memory and mapper, MC45 execution, memory-mapped ports, MCU, RTC, SD card, video (Superfast, Chroma81, sprites, 128/256 characters, 70/80 columns, double buffer, WRX), sound (AY chips, beeper, VGM, PEG and speech), keyboard and joystick, and networking through the WiFi module. Use it to check a freshly assembled interface or to track down a fault. It is in Software.zip, in the SD81/TOOLS/SD81TEST folder: copy the whole folder to the SD card (the five .BIN files, SD81TEST.VGM and SD81TEST.P) and load the program:

```
LOAD “SD81TEST.P”
```

The menu groups the tests by category. Automatic tests end with RESULT: OK or RESULT: FAIL; interactive tests draw or play something and ask whether it looks or sounds right (Y/N).

**Automatic run and report:**

In menu 9 (INFO AND SUMMARY), option 2 runs the 20 automatic tests one after another and shows a summary with the result of each. The summary is also saved to SD81TEST.TXT, in the same folder, together with the date and the MCU, ROM and FPGA versions, to compare machines or attach it to a support request.

The memory test (menu 1, option 1) is destructive: it wipes every RAM page that isn't a system page (CP/M RAM disks, saved screens...). That's why it isn't part of the automatic run. Interactive tests need SLOW mode (the power-on default), and network tests need a configured WiFi module.

# 17. Updating through microSD: firmware, FPGA, ROM and WiFi

## 17.1 Components and file locations

Normal updates use the FAT32 microSD card, without opening the case or connecting a programmer. Four components are involved: STM32 MCU, FPGA, Z80 ROM and the optional ESP32-C3 WiFi module. Use files from the same release package; individual component version numbers are not expected to match each other.

| Component | File and location on the SD card | Version listed in the 1.7 package notes |
|---|---|---|
| STM32 MCU | `/firmware.bin` | 2.7 |
| FPGA | `/sd81.mcs` | 1.6 |
| Z80 ROM | `/SYS/SDBOOST.ROM` | 1.8 |
| Optional ESP32-C3 WiFi | `/ESP32FW.BIN` | 1.4 |
| Debugger monitor | `/SYS/DEBUG.BIN` | Use the matching package |

**Downloads:** project releases. The notes and packaging script identify `Firmware.zip`, `SD Content.zip` (the name may use a dot instead of the space), `Software.zip` and `Tools.zip`. Download the attached assets for your selected release; the automatic "Source code" ZIP is not a substitute for ready-to-use firmware binaries. This edition documents the inspected source revision; it does not certify that every main-branch change has been released as a binary.

## 17.2 Preparation and backup

1. Switch off the ZX81 before removing or inserting the microSD card.
2. Back up the entire card on your computer, including programs, saved games and configuration.
3. Preserve `/SYS/WIFI.CFG`, `/SYS/NTP.CFG` and `/SYS/NET.CFG` if used. The package excludes WIFI.CFG but may include NTP.CFG: check that copying does not replace your settings.
4. Extract the packages. Copy their contents to the locations in the table; do not leave ZIP files or an extra enclosing folder in place of the required files.
5. Keep SYS at the card root. Update SDBOOST.ROM and DEBUG.BIN from the same package as the MCU. Merge the other supplied system files while preserving your configuration.
6. Safely eject the card from the computer and insert it with the ZX81 switched off.

**Do not rename bootloader.bin to firmware.bin.** The bootloader is for advanced recovery, not the normal SD update. Do not use another device's binary or an ESP32 combined initial-flashing image in place of ESP32FW.BIN.

## 17.3 Installing the complete update

1. Switch on the ZX81 with the prepared card inserted.

2. The bootloader finds firmware.bin, programs the MCU and verifies the write. Only after successful verification does it delete the file and start the application.
3. The firmware finds sd81.mcs, programs and verifies the FPGA configuration flash. It deletes this file after success.
4. The ROM is loaded from `/SYS/SDBOOST.ROM` at startup. This file **must remain on the card**; it is not a disposable update installer.
5. If a programmed WiFi module is installed, it looks for ESP32FW.BIN, updates itself and restarts. It deletes the file on success.
6. Wait for startup and update activity to finish. Do not disconnect power or remove the card during programming.

WiFi firmware 1.4 waits up to 120 seconds for the STM32 to respond. Earlier versions wait only about 15 seconds: when MCU and FPGA are updated at the same startup, the WiFi update may remain pending. **After the other updates have finished**, switch off and on again to process ESP32FW.BIN. Do not confuse overall package version 1.7 with WiFi component version 1.4.

## 17.4 Updating WiFi only

With the rest of the system compatible and up to date, copy **ESP32FW.BIN to the microSD root** and restart the system. Copy it using a computer, or upload it through the module's web file server. When uploading over WiFi, wait for the transfer to finish before switching off. Applying an SD update does not require Internet access: the ESP32 receives the image through the STM32.

The old filename was ESP32_FW.BIN. The inspected STM32 firmware maps the old request to the new filename where appropriate, so an older WiFi module can find ESP32FW.BIN. Use the filename supplied by the package and update the STM32 first when upgrading from an old version.

STAT flashes pink while the WiFi update file is being read. An error may leave it yellow. **LED colour alone does not confirm that the complete package is installed:** check versions and diagnostic files.

If the image is invalid or too large, the inspected code deletes the rejected image and writes `/ESP32FWFAILED.TXT` explaining the reason. A transport error during transfer leaves the image on the card for a retry at the next startup. Copy a valid image from the correct package again; disappearance of the BIN file alone is not proof of success.

## 17.5 Verifying the result

- Run `LOAD *VER` to inspect firmware information and `LOAD *FPGA` to inspect the FPGA version.
- Read the WiFi version at the bottom of its web file server or in `/MAN/IP.TXT`. Use `LOAD THEN PRINT "*IP"` to display connection information.
- Check that SDBOOST.ROM and DEBUG.BIN remain in SYS. In package 1.7, `LOAD *Z81` requires the DEBUG.BIN monitor; without it, the command returns report P.
- Test a program, video and sound. For broader diagnostics use SD81TEST (section 16.8), whose report records versions. Its memory test destroys RAM contents; do not run it with unsaved work.
- If WiFi still shows the old version after the other updates, perform the second startup described above and inspect ESP32FWFAILED.TXT if present.

## 17.6 Errors and recovery

| Symptom | Check |
|---|---|
| MCU: alternating blue/red | Bootloader SD initialization failed. Check FAT32 card and contacts. |
| MCU: alternating white/red | Programming or verification failed. The image remains for a retry with a valid copy. |

| Symptom | Check |
|---|---|
| Yellow during programming | May indicate MCU writing or a WiFi error. Identify the phase and consult diagnostics without interrupting an active write. |
| WiFi: BIN gone but version unchanged | Inspect ESP32FWFAILED.TXT; the image may have been rejected. |
| Commands missing or expansion fails to start | Check SYS, ROM and package consistency before attempting USB recovery. |

Initial programming of a blank MCU or ESP32, and recovery when SD loading cannot run, are separate operations. The repository describes STM32 recovery using JP7 and STM32CubeProgrammer, and ESP32-C3 programming using Arduino. The packaging script places recovery instructions and tools in **Tools.zip**. Consult the instructions for the specific version or technical support before opening the case; normal updates in this guide do not require it.

**RTC:** normal firmware updates do not require clock initialization. Reserve initialization for first programming or corrupted configuration; do not hold QuickSilva at startup as part of a routine update.

# 18. Glossary

| Term | Definition |
|---|---|
| **Allophone** | Minimal speech sound unit used by the speech synthesiser. The SD81 Booster uses the SP0256 chip allophones. |
| **Block** | 8 KB division of the Z80 address space. The SD81 Booster divides the Z80’s 64 KB into 8 blocks (0-7). |
| **FPGA** | Programmable logic circuit (Xilinx Spartan-6 XC6SLX9) that implements in hardware the video logic, memory mapper and other interface functions. |
| **FAT32** | Filesystem required by the interface’s microSD card. Incompatible with exFAT and NTFS. |
| **FAST** | ZX81 BASIC token (SHIFT+F). On the SD81 Booster, activates SD loading/saving mode. |
| **Screen file (HFILE)** | Memory block containing the screen data in Superfast modes. |
| **HRG** | High Resolution Graphics. ZX81 high resolution mode. |
| **MCU** | Microcontroller. The chip that manages the SD, sound, RTC and communication with the Z80. |
| **Page** | 8 KB division of the interface RAM. The 512 KB are divided into 64 pages (0–63) that can be mapped to any block. |
| **PEG** | Programmable Effects Generator. Virtual machine for playing sound effects in the background without using the ZX81 CPU. |
| **RTC** | Real Time Clock. Real time clock incorporated in the SD81 Booster, powered by a CR2032 battery. |
| **SLOW** | ZX81 BASIC token (SHIFT+D). On the SD81 Booster, activates tape (audio) loading/saving mode. |
| **SP0256** | General Instrument speech synthesiser chip, base of the SD81 Booster synthesiser. Also used in the Currah MicroSpeech and The Voice. |
| **Superfast** | Mode where the SD81 Booster hardware manages screen refresh, freeing the ZX81 CPU for other tasks. |
| **T81** | Container file format that groups multiple ZX81 programs. The interface can navigate its content as if it were a directory. (alpha stage) |
| **Token** | In ZX81 BASIC, each reserved word is stored as a single byte. Entered with SHIFT key combinations. |

| Term | Definition |
|---|---|
| **VGM** | Video Game Music. File format for music that the SD81 Booster can play in the background using the AY emulator. |
| **Inverse video** | ZX81 display mode where background and character swap colours. Activated with SHIFT+9 before the character. |

# 19. Firmware Version History

| Version | Date | Main changes |
|---|---|---|
| **1.0** | 2025 | First public release. |
| **1.1** | 07/2026 | Update check and install from the WiFi module’s web page; module diagnostic log on the SD card. |
| **1.2** | 09/2026 | Hardware sprites, Superfast fine scroll, 256 definable characters; Chroma81 colour and joystick fixes. |
| **1.3** | 09/2026 | Modem-style network bridge (BBS/telnet, AT commands), 70 and 80 column modes, MC45 in blocks 6 and 7, alternative screen and attributes, and CP/M 3. |
| **1.4** | 09/2026 | EightyOne snapshots (LOAD *Z81), LOAD *ROMLOCK and fix for border garbage with WRX games. |
| **1.5** | 09/2026 | TELNET terminal and SD81TEST test program. FPGA: synchronous write capture and FRAMES reads in Superfast. MCU: DEL with absolute paths and reading AY registers 14 and 15. |
| **1.6** | 10/2026 | Hardware debugger (Appendix L): on-screen display on the ZX81, symbols, trace and history, console and panels in the browser; full snapshots with LOAD *Z81. Simulated interrupts in the FPGA. ROM: PLOT and TAB in the 70- and 80-column modes (thanks to Kelly Abrantes Murta for reporting the PLOT bug). MCU: sound is silenced while a snapshot loads; the WiFi module web server is smoother. New Appendix K with the EightyOne emulator guide. |
| **1.7** | 10/2026 | Network turn-based games (section 9.9 and Appendix M): Connect 4 and chess with clocks, drawn in high resolution, which save the game to the SD card and can be carried on another day. WiFi module: MQTT, SHA-256, HMAC and random-number AT commands. 64 sprites, up to 12 per line. Debugger: screen capture and ZX81 keyboard in the web page, sym !, and CP/M. LOAD *Z81 only through the debugger monitor (report P; the MCU command 70 is gone). The WiFi module’s update file is now ESP32FW.BIN and everything can be updated in one go. 32-bit WAV files play correctly. Versions: STM32 2.7, ROM 1.8, FPGA 1.6, WiFi 1.4. |

| INFO | **This history will be updated with each new firmware version. Check the project repository for the complete changelog.** |
|---|---|

# 20. References

## The SD81 Booster Project

**Official repository** — source code, firmware, schematics and technical documentation:

https://github.com/wilco2009/SD81-Booster

**Va de Retro** — Spanish retrocomputing forum where a previous version of the interface was announced:

https://www.va-de-retro.com/foros/portal

## Tools

**STM32CubeProgrammer** — STM32 MCU programming tool, needed for USB recovery in emergencies:

https://www.st.com/en/development-tools/stm32cubeprog.html

## Technical Reference Documentation

**SP0256-AL2 speech synthesiser chip** (General Instrument) — datasheet for the chip on which the SD81 Booster speech synthesiser is based:

https://rarewaves.net/wp-content/uploads/2018/09/SP0256-AL2.pdf

**Chroma81 interface** — documentation for the ZX81 colour interface whose functionality the SD81 Booster implements. The original link is no longer available; it can be found in web archives:

http://www.fruitcake.plus.com/Sinclair/ZX81/Chroma/ChromaInterface_Documentation.htm

**.P and .P81 file format** — technical specification of the ZX81 program formats:

https://k1.spdns.de/Develop/Projects/zasm/Info/O80%20and%20P81%20Format.txt

**VGM format specification** (Video Game Music) — file format for music used by the interface VGM player:

https://vgmrips.net/wiki/VGM_Specification

## ZX81 ROM

The SD81 Booster includes a modified ROM based on the original ZX81 disassembly. Credits:

**Geoff Wearmouth** — commented ZX81 ROM disassembly (preserved at archive.org):

https://web.archive.org/web/20150815035607/http://www.wearmouth.demon.co.uk/zx81.htm

**Tomaž Šolc** — preservation of the disassembly:

https://www.tablix.org/~avian/spectrum/rom/

## Project Credits

- Hardware design and MCU firmware: Alejandro Valero (wilco2009)
- Z80 code / Modified ROM: Pedro Gimeno (pgimeno)

# Appendix A — Complete PLAY Command Reference

## Duration Table

| Value | Name | Duration at 60 bpm |
|---|---|---|
| **1** | Semiquaver (1/4 crotchet) | 0.25 s |
| **2** | Dotted semiquaver | 0.375 s |
| **3** | Quaver (1/2 crotchet) | 0.5 s |

| Value | Name | Duration at 60 bpm |
|---|---|---|
| **4** | Dotted quaver | 0.75 s |
| **5** | Crotchet | 1 s |
| **6** | Dotted crotchet | 1.5 s |
| **7** | Minim (2 crotchets) | 2 s |
| **8** | Dotted minim | 3 s |
| **9** | Semibreve (4 crotchets) | 4 s |
| **10** | Semiquaver triplet | 0.1667 s |
| **11** | Quaver triplet | 0.3333 s |
| **12** | Crotchet triplet | 0.6667 s |

## Envelope Table (W)

| Code | Shape | Description |
|---|---|---|
| **W0** | \|_____ | Decay then stay off |
| **W1** | /|_____ | Rise then stay off |
| **W2** | \|¯¯¯¯¯ | Decay then stay on |
| **W3** | /¯¯¯¯¯ | Rise then stay on |
| **W4** | \|\|\|\|\| | Repeated decay |
| **W5** | /|/|/|/| | Repeated rise |
| **W6** | /\/\/ | Rise then decay, repeat |
| **W7** | \/\/\ | Decay then rise, repeat |

## Complete Command Table

| Parameter | Description |
|---|---|
| C..B | Note in current octave |
| Inv(C..B) | Note in next octave (inverse video) |
| = | Sharpens the next note |
| £ | Plays a rest |
| 1..12 | Duration from this point |
| - | Duration ligature |
| N / space | Number separator |
| O<n> | Octave (0–8, default 4) |
| T<n> | Tempo in bpm (60-240, default 120) — channel A only |
| V<n> | Volume (0–15) |

| Parameter | Description |
|---|---|
| W<n> | Selects envelope effect (0–7). |
| U | Enables envelope for the channel. |
| X<n> | Sets envelope ramp time (0–65535; 6927 ≈ 1 second). |
| M<n> | Active channel and mode selection (0–63) |
| ( ) | Repeat enclosed section once more |
| ) | Repeat from start indefinitely |
| H | Forces termination of the entire PLAY command. |
| * | (Channel A only) Background playback |

# Appendix B — PEG Effects Generator Reference

The PEG is a 16-bit virtual machine with access to the 16 AY chip registers (R0–R15) and 16 general purpose variables (V0–V15). It supports up to 3 parallel execution threads and up to 256 16-bit words of program memory.

| Instruction | Encoding | Description |
|---|---|---|
| LD R,XX | 0R XX | Load register with 8-bit value |
| ADD R,XX | 1R XX | Add 8-bit value to register |
| LD V,XX | 2R XX | Load variable with 8-bit value (rest zeros) |
| ADD V,XX | 3R XX | Add 8-bit value to variable |
| LD R,R | 40 RR | Load register from another register |
| LD R,V | 41 RV | Load register from variable |
| LD V,R | 42 VR | Load variable from register |
| LD V,V | 43 VV | Load variable from another variable |
| ADD V,V | 44 VV | Add second variable to first |
| SUB V,V | 45 VV | Subtract second variable from first |
| ADC V,V | 46 VV | Add with carry |
| SBC V,V | 47 VV | Subtract with carry |
| NOT V,V | 48 VV | First var = bitwise NOT of second |
| AND V,V | 49 VV | Bitwise AND |
| OR V,V | 4A VV | Bitwise OR |
| XOR V,V | 4B VV | Bitwise XOR |
| MUL V,V | 4C VV | Multiply (32-bit result split across two vars) |
| DIV V,V | 4D VV | Divide (quotient + remainder in next var) |

| Instruction | Encoding | Description |
|---|---|---|
| SHR V,X | 4E VX | Shift right |
| SHL V,X | 4F VX | Shift left |
| MUL V,XX | 5V XX | Multiply variable by constant |
| DIV V,XX | 6V XX | Divide variable by constant |
| SUB V,XX | 7V XX | Subtract constant from variable |
| DJNZ V,XX | 8V XX | Decrement and jump by offset if not zero |
| WAIT XXX | 9X XX | Wait given milliseconds |
| WAIT V | A0 0V | Wait time specified in variable |
| HALT | A0 10 | Stop effect |
| JR XX | A1 XX | Jump by offset |

| INFO | Jump offsets are relative to the next instruction. The PEG assembler included in the repository handles this automatically. |
|---|---|

# Appendix C — Speech Synthesiser Dictionary

The synthesiser is based on the SP0256 phonemes from General Instrument (Currah MicroSpeech, The Voice). Text is analysed left to right looking for the longest possible match in the dictionary. Recognised words sound better than letter-by-letter phonetic decomposition.

## Recognised Words (selection by length)

13-character words: INVESTIGATORS, IRRESPONSIBLE

12-character words: INVESTIGATOR

11-character words: INVESTIGATE

10-character words: CORRECTING

9-character words: COGNITIVE, CORRECTED, SEPTEMBER, SINCERELY, SINCERITY, INTERFACE

8-character words: CHECKERS, CHECKING, COMPUTER, CORRECTS, DAUGHTER, DECEMBER, EIGHTEEN, FEBRUARY, FREEZERS, FREEZING, NINETEEN, NOVEMBER, PLEDGING, SATURDAY

7-character words: BOOSTER, CHECKED, CHECKER, CORRECT, FREEZER, JANUARY, MINUTES, OCTOBER, PLASTIC, SIXTEEN, TUESDAY, COLLIDE

6-character words: AUGUST, COOKIE, EQUALS, EXTENT, FRIDAY, FROZEN, MONDAY, SUNDAY, TALKED, TALKER, TWENTY

5-character words: APRIL, CHECK, CROWN, EIGHT, EQUAL, ERROR, FIFTY, HELLO, MARCH, MONTH, SIXTY, TALKS, THREE, WORLD

4-character words: DATE, FIVE, FOUR, HAVE, JUNE, NINE, RAYS, TALK, THIS, TIME, WHAT, WHOA, WILL, ZX81

3-character words: ACK, ACT, ADD, AMP, ASH, ASK, BAD, BED, BIG, BOX, BUT, CAR, END, EST, GET, HAS, HIM, ICK, IMP, ING, INK, JOB, KEY, MAY, NOT, NOW, OLD, OUR, OUT, RAY, RED, SIX, SUN, TEN, THE, TOP, TWO, USE, WAY, WHO, WHY, YES, YET, YOU, and many more.

### Punctuation and Pauses

| Character | Effect |
|---|---|
| Space | Short pause |
| , | Medium pause |
| ; : | Long Pause |

### Direct Allophones (advanced use)

Pauses: PA1, PA2, PA3, PA4, PA5

Vowels: AA, AE, AH, AO, AW, AX, AY, EH, ER1, ER2, EY, IH, IY, OW, OY, UH, UW1, UW2, XR, YR

Consonants: BB1, BB2, CH, DD1, DD2, DH1, DH2, EL, FF, GG1, GG2, GG3, HH1, HH2, JH, KK1, KK2, KK3, LL, MM, NG, NN1, NN2, OR, PP, RR1, RR2, SH, SS, TH, TT1, TT2, VV, WH, WW, YY1, YY2, ZH, ZZ

# Appendix D — Memory Paging System

### Initial Page Assignment

| Block | Default page | Range / Use |
|---|---|---|
| **0** | 0 | 0000–1FFF (ROM, read only) |
| **1** | 1 | 2000–3FFF (ROM Extension) |
| **2** | 2 | 4000–5FFF (Main RAM) |
| **3** | 3 | 6000–7FFF (Main RAM) |
| **4** | 4 | 8000–9FFF (Extra RAM) |
| **5** | 5 | A000–BFFF (Extra RAM) |
| **6** | 2 | C000–DFFF (Block 2 mirror) |
| **7** | 3 | E000–FFFF (Block 3 mirror) |

### Use Rules

- Block 0 is always read only. Blocks 1–7 are read/write.
- The same page can be mapped to more than one block simultaneously.
- Blocks 6 and 7 must mirror blocks 2 and 3 respectively for the video system to work. Exception: if the display file is entirely in block 2, block 7 can be freely mapped; if entirely in block 3, block 6 can be freely mapped.
- Blocks 4 and 5 can always be mapped to any page.

- Due to ZX81 hardware quirks, blocks 4–7 can only be used for data, not to execute code (except with MC45 mode active for blocks 4 and 5).

## ROM Modification

It is possible to modify the ROM by mapping page 0 to any write-enabled block and making changes there. The ROM can also be completely replaced by loading new content into a page and mapping it to block 0.

**Machine Code Execution Restriction**

By default, any instruction in blocks 4–7 whose opcode has bit 6 equal to 0 (opcodes 0–63 and 128–191) is replaced by a NOP by the ZX81 hardware. MC45 mode (section 11.3) removes this restriction for blocks 4 and 5.

**Simple vs. Full Paging Mode**

**Simple mode (up to 256 KB):** uses bits D2–D0 of port E7h for the block and bits D7–D3 for the page (0–31).

**Full mode (up to 512 KB):** uses the instruction OUT (C),r with the page in B (0–63). Changed via the FULLPAGING/HALFPAGING MCU commands.

# Appendix E — Chroma81 Interface Port (7FEFh)

The SD81 Booster implements the Chroma81 colour interface through port 7FEFh (01111111 11101111 in binary). This port can be read and written.

## Writing (OUT 7FEFh)

Allows you to set the color mode and border color:

| Bits | Function |
|---|---|
| **7–6** | Reserved for future use. Always at 0. |
| **5** | Activate color mode: 1 = color on. |
| **4** | Colour mode: 0 = character code, 1 = attribute file. |
| **3** | Border color brightness bit. |
| **2–0** | Border color in GRB (Green-Red-Blue) format. |

### Border colour format (bits 2-0, GRB format)

| Value (GRB) | Color |
|---|---|
| **000** | Black |
| **001** | Blue |
| **010** | Red |
| **011** | Magenta |
| **100** | Green |

| Value (GRB) | Color |
|---|---|
| **101** | Cyan |
| **110** | Yellow |
| **111** | White |

## Reading (IN 7FEFh)

Allows detecting whether colour mode is available and reading VSync status:

| Bit | Function |
|---|---|
| **7–6** | Not used (reserved). |
| **5** | 0 = colour modes available. Always 0, indicating colour mode is always active. |
| **4–1** | Not used (reserved). |
| **0** | VSync interrupt status. 1 = the electron beam is in the vertical blanking period (screen painted). |

| INFO | **Reading bit 5 as 0 allows programs to automatically detect the presence of the Chroma81 interface and activate colour modes if available.** |
|---|---|

Equivalent command:

LOAD *COLOR : REM enable colour, border white (7) by default

LOAD *COLOR <colour> : REM enable colour, border = <colour> (0-15)

LOAD *COLOR <colour>,<mode> : REM colour = <colour> (0-15), Chroma mode = <mode> (0-1)

LOAD *COLOR STOP : REM disable colour

(equivalent to the OUT 7FEFh above; sets bit 5 and the character-code mode automatically. **<colour>** is the port's low 4 bits, brightness + GRB combined into 0-15. **<mode>** is optional (0 = character-code colour table, 1 = attribute file; 0 if omitted). Mode 1 is only implemented in hardware for the Superfast submodes: in native mode (Superfast off) it has no real effect.

**VSync synchronisation:**

Bit 0 allows the CPU to wait until the screen has finished painting before updating its content, avoiding flicker and visual artefacts. Many ZX Spectrum games used the HALT instruction or an interrupt routine to synchronise with VSync; on the SD81 Booster this mechanism is the direct equivalent for that functionality:

```
; Wait for start of VSync

WAIT: in a,($AF)

rrca ; bit 0 to carry

jr nc,WAIT ; if carry=0, screen still painting

; screen refresh complete, safe to update video
```

# Appendix F — Superfast and Spectrum Modes

## The Video Problem in the Original ZX81

The ZX81 in SLOW mode manages video by software: during screen refresh, the CPU must execute NOP instructions while the hardware generates the video signal, consuming a significant fraction of processor time.

## Superfast Mode

In Superfast mode, the interface hardware takes control of the data bus during screen refresh, placing video data directly without CPU intervention. This frees the processor completely for other tasks.

Before activating the mode it is necessary to indicate the screen file address (HFILE):

> POKE 2043, HFILE_low : REM low byte of screen file address
>
> POKE 2044, HFILE_high : REM high byte of screen file address

The three Superfast mode variants are selected by POKEing address 2045:

| POKE | Mode |
|---|---|
| POKE 2045, 170 | Superfast text (standard text mode, accelerated) |
| POKE 2045, 171 | Superfast HiRes native (high resolution in ZX81 format) |
| POKE 2045, 172 | Superfast HiRes Spectrum (high resolution in Spectrum format) |
| POKE 2045, 85 | Deactivate Superfast mode |

Equivalent commands:

LOAD *SFAST : REM Superfast text mode

LOAD *SFHR <address> : REM Superfast native HiRes, HFILE = <address>

LOAD *SFSP <address> : REM Superfast Spectrum HiRes, HFILE = <address>

LOAD *SFAST STOP : REM disable (equivalent to *SFHR STOP / *SFSP STOP)

(equivalent to the POKEs above, for machine-code use). **LOAD *SFHR** and **LOAD *SFSP** write HFILE (2043/2044) and enable the mode in a single step; HFILE is not needed for **LOAD *SFAST** since text mode does not use extended RAM.

## Fine Horizontal Scroll (Superfast)

All three Superfast submodes support a fine horizontal scroll offset of 0 to 7 pixels, which advances the character/attribute/pixel fetch sequence by that many pixels on every row:

POKE 2090, <offset> : REM fine scroll offset, 0-7 pixels

Equivalent command:

LOAD *SCROLL <offset> : REM equivalent to POKE 2090

The offset applies equally to all three Superfast submodes (text, native HiRes and Spectrum HiRes); it has no effect in native mode. In Superfast text mode, the pixel gap left at the right edge of each row is filled by the DFILE's NEWLINE byte (position 32 of each row, unused in this mode): software is responsible for writing whatever content should appear there as the scroll advances. This happens whether or not colour is enabled (the character fetch itself does not depend on Chroma); the only thing that can vary with the Chroma mode is the colour this extra column is shown in, and only in mode 1 (attribute file), which does not compute this column's colour correctly. In the other submodes (native HiRes and Spectrum HiRes) that gap does not have a defined content yet.

The scroll can be turned on or off per row, to keep markers or a score line fixed while the rest of the screen scrolls. Each text row (0-23) has one bit in one of three 8-bit registers:

POKE 2091, <map> : REM rows 0-7 (bit0=row 0)

POKE 2092, <map> : REM rows 8-15

POKE 2093, <map> : REM rows 16-23

Bit=1 means that row is shifted by the **LOAD *SCROLL** offset; bit=0 keeps that row fixed. By default (after a reset) all three registers are 255 (every row scrolls), matching the behaviour before this register existed. Equivalent command:

LOAD *SCROWS <lo>,<mid>,<hi> : REM equivalent to POKE 2091/2092/2093

(equivalent to the three POKEs above in a single step; each argument is a plain byte, 0-255).

## Alternative Screen Address (Superfast)

The Superfast text submodes (32, 70 and 80 columns) normally take the screen address from the D_FILE system variable (16396/16397), just like native mode. The SD81 Booster lets you replace that address with another one, without touching D_FILE, via an alternative register:

> POKE 2096, <addr_low> : REM low byte of the alternative address
>
> POKE 2097, <addr_high> : REM high byte
>
> POKE 2098, 170 : REM enables the alternative register
>
> POKE 2098, 85 : REM disables it, back to D_FILE

While the alternative register is disabled (the default, both right after a reset and after POKE 2045,85), the behaviour is exactly the same as before: nothing changes for existing software. Once enabled with POKE 2098,170, the Superfast screen is read from the address given in 2096/2097 instead of D_FILE, without BASIC or the ROM routines (which keep using D_FILE for their own purposes) noticing the change.

This makes it possible, for example, to keep two screens in memory and switch which one is displayed without having to move D_FILE back and forth every frame.

**Important:**

Returning to native video mode (POKE 2045,85) automatically disables the alternative register, so that "standard mode" always means the usual behaviour. Switching between the different Superfast submodes (POKE 2045,170/171/172/173/174) does not disable it: if it was on, it stays on.

## Alternative Attribute Table (Chroma Mode 1, Superfast)

Normally, Chroma mode 1 (attribute file) places the colour table at the same address as the screen but with bit 15 forced to 1, i.e. $8000 away from D_FILE (or the alternative address from the previous section, if enabled). The SD81 Booster lets you decouple this table too, with the same mechanism:

> POKE 2059, <base_low> : REM low byte of the attribute table

> POKE 2060, <base_high> : REM high byte
>
> POKE 2061, 170 : REM enables the alternative table
>
> POKE 2061, 85 : REM disables it, back to the usual behaviour

Disabled (the default), mode 1 keeps working exactly as before. Enabled with POKE 2061,170, the colour table is no longer tied to the screen position: it can live at any 16-bit address, without the modulo-32K limit that forcing bit 15 imposed. The table uses the same row/column arithmetic as the screen (including the initial padding byte), so software can reuse the same position calculation for both.

This register is independent from the one in the previous section: you can enable the alternative screen address without enabling this one, or the other way round, though normally both will be used together for full control over where each thing lives.

**Important:**

Just like the alternative screen address, returning to native video mode (POKE 2045,85) also disables this register automatically.

## Spectrum Mode

Spectrum mode (POKE 2045,172) reorders screen lines to match the ZX Spectrum screen organisation, facilitating the conversion of programs between both platforms and enabling the loading of .SCR files directly.

| NOTE | **In Spectrum mode, the Spectrum beeper and edge port (ULA write port = FBh) is emulated, where bits 2–0 control the edge color and bits 4–3 control the beeper. In this mode, the FBh port is occupied and compatibility with the ZX Printer is lost.** |
|---|---|

## Border control

**Change border attributes:**

> POKE 2046, <attr>

**Define and activate a border pattern:**

It is possible to define an 8-byte pattern that will be repeated along the screen border, allowing it to be completely filled with a custom design:

> POKE 2048, byte0 : REM first byte of pattern
>
> POKE 2049, byte1
>
> ...
>
> POKE 2055, byte7 : REM last byte of pattern
>
> POKE 2047, 170 : REM activate border pattern
>
> POKE 2047, 85 : REM deactivate border pattern

**ULA port in Spectrum mode (FBh):**

| Bits | Function |
|---|---|
| **4–3** | Beeper control |
| **2–0** | Border color |

Equivalent command (pattern ink + Spectrum-mode border):

LOAD *BORDER <colour> : REM pattern ink (0-15) and Spectrum-mode border (bits 2-0)

(equivalent to **POKE 2046,<colour>** plus the ULA port FBh above in a single step; it does not enable the pattern by itself, **POKE 2047,170** is still needed. The native-mode border background colour is set with **LOAD *COLOR**, not this command.)

## VSYNC Synchronisation

On the ZX Spectrum, many games used the HALT instruction or an IM1/IM2 interrupt routine to synchronise with the vertical screen scan and achieve smooth animations without flickering.

On the SD81 Booster this functionality is replaced by reading bit 0 of port AFh, which reflects the status of the vertical sync interrupt (VSYNC). The CPU can wait for this bit to synchronise with the start of screen refresh without needing interrupts or HALT.
Bits 6..1 are the VSYNC counter since the last read and bit 0 on read indicates the instantaneous state of VSYNC)

## Double buffering (present-blit)

### How the interface generates the image (without double buffering)

All of the system’s RAM (up to 512 KB) lives in a single external memory chip (the **SRAM**), organised in 8 KB pages that get assigned to the Z80’s 8 memory blocks through the mapper (E7h). The video circuit, however, does **not** read that SRAM directly — doing so would make it compete with the CPU for the same memory chip on every cycle. To avoid that, the interface keeps an **internal video memory** (64 KB) inside the FPGA itself, acting as a **mirror** of the 8 blocks: every time the CPU writes a byte to the SRAM, that same byte is automatically copied into the mirror. The video circuit always generates the image by reading this internal mirror, never the SRAM.

The mirror is faithful at all times — too faithful, in fact: if at some instant the CPU has just erased a sprite, the mirror already reflects the empty gap, and if the video beam happens to scan that area right then, it draws that gap. Because the CPU and the video circuit share the same “live” data, the beam can catch intermediate drawing states — hence the flicker and tearing that double buffering is meant to fix.

### What double buffering changes

POKE 2057, 168+B : REM enable double buffer; front buffer = block B (0-7)

POKE 2057, 85 : REM disable double buffer

When enabled, the interface stops keeping the real-time mirror **only for block B** of the internal video memory (every other block of the mirror keeps working as before). From that point on:

- The video circuit stops looking at the HFILE block’s mirror (the usual screen) and starts looking at block B’s mirror instead.

- Block B’s mirror is **no longer updated write by write**. On every vertical blanking (VSYNC), the hardware copies, byte by byte, the entire current content of the HFILE block’s mirror into block B’s mirror in one go. Between one VSYNC and the next, block B’s mirror stays **frozen**: it is a fixed snapshot, not a live reflection.

> **The part that trips people up — read it twice:** logical block B still exists as normal SRAM too, just as accessible and writable by the CPU as any other block. But while double buffering is active, **that SRAM has no relation whatsoever to what you see on screen** — it is that block’s internal mirror that stops reflecting it. Whatever you write to block B’s SRAM will not show up in the image, and what you see on screen is not that SRAM but the snapshot the FPGA refreshed at the last VSYNC, copied from the HFILE block. That is why block B should be treated as “reserved for the hardware” while double buffering is active, even though you can technically still use it as RAM for unrelated things (variables, code) that have nothing to do with the screen.

Put differently, three different things coexist under the same block number, and they should not be confused:

| Layer | What it contains | Who updates it |
|---|---|---|
| Physical SRAM of block B | The system's real memory (8 KB) | The CPU, on every normal read/write |
| Internal mirror of block B — double buffer OFF | A live copy of that SRAM | Automatically, byte by byte, on every CPU write |
| Internal mirror of block B — double buffer ON | A frozen snapshot of the HFILE block | Only the FPGA, all at once, once per VSYNC |

The practical result:

- The screen always shows a **complete snapshot** taken at the last VSYNC: erases or half-drawn sprites are never visible.

- The program always draws on **a single surface** (the usual HFILE page): no page swapping, no "redraw what changed two frames ago" bookkeeping.

- Reading back from the screen (the HFILE page, not block `B`) always returns the last value written — coherent read-modify-write.

**Correct usage:** wait for the rising edge of VSYNC (bit 0 of port AFh) and do all erasing/drawing right after. From that moment you have about **16 ms** before the hardware takes the next snapshot (the copy starts right after the visible area ends). If drawing takes longer, the snapshot may capture an intermediate state — the same behaviour as a classic double buffer.

**Choosing the front block (`B`):** since that block's mirror stops reflecting its SRAM, **no video element** (HFILE, DFILE, Chroma81 attributes) must point to it while double buffering is active — its mirror would no longer be able to display them. Recommended blocks: **4 or 5** ($8000-$9FFF / $A000-$BFFF, whichever does not hold your HFILE). Avoid: 0 (text-mode ROM glyphs), 1 (chr RAM), 2-3 (DFILE), 6-7 (Chroma81 attributes). If the front block matches HFILE's block, the display freezes permanently: the blit just copies the block onto itself every VSYNC (no effect), and the write mask permanently blocks your new drawing from ever reaching the mirror --- the SRAM does get updated, but none of it ever becomes visible.

> **Note:** in native HiRes mode only the bitmap is double-buffered; attributes (the $C000 area) are read live. In Spectrum mode the whole block is double-buffered (bitmap + attributes). Text mode does not use the double buffer.

**Typical setup** (HFILE at $8000 = block 4, front at block 5):

POKE 2043,0 : REM HFILE low

POKE 2044,128 : REM HFILE high ($8000)

POKE 2045,172 : REM Superfast Spectrum HiRes

POKE 2057,173 : REM double buffer ON, front = block 5 (168+5)

Equivalent command:

LOAD *DBUF 5 : REM double buffer ON, front = block 5

LOAD *DBUF STOP : REM double buffer OFF

**I/O port control (pseudo-block 8):** `POKE 2057` stops working once a program has disabled the control-POKE window by writing to 2056 ("flat RAM" mode, e.g. CP/M). For those cases the double buffer can also be controlled through the **mapper port (E7h)** using the fictitious block 8:

asm

ld a,08h ; pseudo-block 8

ld b,32+5 ; value: bit5=enable, bits2:0=front block (here 5)

ld c,0e7h

out (c),a ; double buffer ON, front = block 5

ld b,0 ; value 0 = disable

out (c),a

This path is available when **full paging** mode is active or after writing to 2056. Restriction: in half paging after 2056, do not assign an odd page to block 0 through the mapper port (that data pattern, x8h, matches pseudo-block 8).

**MANUAL mode: double buffering without an automatic copy**

In addition to the AUTO mode described above (with an automatic blit on every VSYNC), there is a **MANUAL mode** where the FPGA copies nothing: the Z80 itself draws the whole frame directly into whichever block is not currently the front, and only switches which of the two blocks is shown. This saves the cost of the automatic copy (~630 µs per frame), at the price of the Z80 having to redraw the whole frame itself (workable in Superfast modes, where the CPU is free from video refresh). Unlike AUTO mode, there is no single HFILE that the FPGA copies here: you need two real blocks (for example 4 and 5).

POKE 2057, 200+B : REM double buffer ON, MANUAL mode; front = block B (0-7)

The values 168+B (AUTO mode) and 200+B (MANUAL mode) share the same POKE 2057; use 85 to disable in either case.

**Typical use:**

1. Pick two real blocks for the two buffers (e.g. 4 and 5). HFILE is irrelevant to this mechanism while double buffering is active.

2. Enable it with the starting block as front (POKE 2057, 200+4) and draw the first frame into the other block (5).

3. Every frame: draw the whole frame into whichever block is not the current front; wait for VSYNC (bit 0 of port AFh); switch the front with POKE 2057, 200+B (B = the block you just drew into).

4. On the next frame, draw into the block that has just been freed (the former front).

The FPGA's write mask still automatically protects the current front block from CPU writes, in both modes.

**I/O port control (pseudo-block 8):** same as in AUTO mode, but with an extra bit (bit4 of the B value) to select MANUAL:

ld a,08h ; pseudo-block 8

ld b,32+16+5 ; bit5=enable, bit4=MANUAL mode, bits2:0=front block (here 5)

ld c,0e7h

out (c),a ; double buffer ON, MANUAL mode, front = block 5

See the complete machine-code example in `EXAMPLES/DBUF/` (bouncing ball with real-time double-buffer toggling).

## WRX with the 8-16K RAM

Some WRX high-resolution programs place their graphics in the **8-16K** area ($2000-$3FFF) and point the I register there (for example, Psion's **Hi-res Chess**). On a real ZX81 this requires a RAM expansion in that region; on the SD81 Booster that region exists, but by default the interface treats it as a **RAM character**

**generator** (the LOAD *128C feature and games with user-defined characters), which needs exactly the opposite behaviour during the video refresh. Both are legitimate uses of the same region and cannot be told apart automatically, so it is selected with the command:

LOAD *WRX : REM enable WRX mode in 8-16K

LOAD *WRX STOP : REM disable (character generator mode, default)

(equivalent to POKE 2058,170 / POKE 2058,85, for machine-code use). Enable WRX mode **before loading** the program (for example LOAD *WRX and then LOAD “HRCHESS”). After a reset it returns to the default mode. It does not affect WRX programs that place their graphics at $4000 or above, which always work without this command.

## Control POKEs summary

| Address | Value | Function |
|---|---|---|
| 2043 | <low> | Low byte of screen file address |
| 2044 | <hi> | High byte of screen file address |
| 2045 | 170 | Activate Superfast text mode |
| 2045 | 171 | Activate Superfast native HiRes |
| 2045 | 172 | Activate Superfast Spectrum HiRes |
| 2045 | 85 | Deactivate Superfast |
| 2046 | <attr> | Change border attributes |
| 2047 | 170 | Activate border pattern |
| 2047 | 85 | Deactivate border pattern |
| 2048–2055 | <data> | Define border pattern (8 bytes) |
| 2056 | xxxx | Disable control pokes and enable writing to block 0 |
| 2057 | 168+B | Enable double buffer (front buffer = block B, 0-7) |
| 2057 | 200+B | Enable double buffer, MANUAL mode (front buffer = block B, 0-7) |
| 2057 | 85 | Disable double buffer |
| 2058 | 170 | Enable WRX in the 8-16K RAM |
| 2058 | 85 | Disable WRX (character generator mode) |
| 2090 | 0-7 | Fine horizontal scroll, Superfast (0-7 pixels) |
| 2091 | <map> | Rows 0-7 with fine scroll active (bit0=row 0) |
| 2092 | <map> | Rows 8-15 with fine scroll active |
| 2093 | <map> | Rows 16-23 with fine scroll active |
| 2096 | <addr_low> | Low byte of the alternative screen address (Superfast) |
| 2097 | <addr_high> | High byte of the alternative screen address |
| 2098 | 170 / 85 | Enable / disable the alternative screen address |
| 2059 | <base_low> | Low byte of the alternative attribute table (Chroma mode 1) |
| 2060 | <base_high> | High byte of the alternative attribute table |

| Address | Value | Function |
|---|---|---|
| 2061 | 170 / 85 | Enable / disable the alternative attribute table |
| 2062 | 170 / 85 | Extend MC45 to blocks 6 and 7 as well / back to 4 and 5 only |

With LOAD *ROMLOCK active, writes to every address in this table and to the sprite registers (2100-2128, Appendix H) have no effect, as with a real ROM. LOAD *ROMLOCK STOP enables them again (see 7.5).

# Appendix G — Audio Technical Reference: AY chip, VGM and allophones

## AY-3-8910/12 Chip Registers

The SD81 Booster AY emulator is register-level compatible with the original chip. It supports three independent voices, envelope and noise.

| Reg | Description | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|---|
| **R0** | Channel A — tone period (low) | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| **R1** | Channel A — tone period (high) | — | — | — | — | B3 | B2 | B1 | B0 |
| **R2** | Channel B — tone period (low) | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| **R3** | Channel B — tone period (high) | — | — | — | — | B3 | B2 | B1 | B0 |
| **R4** | Channel C — tone period (low) | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| **R5** | Channel C — tone period (high) | — | — | — | — | B3 | B2 | B1 | B0 |
| **R6** | Noise period | — | — | — | B4 | B3 | B2 | B1 | B0 |
| **R7** | Channel enable | — | — | Noise C | Noise B | Noise A | Tone C | Tone B | Tone A |
| **R8** | Channel A amplitude | — | — | — | Env. | L3 | L2 | L1 | L0 |
| **R9** | Channel B amplitude | — | — | — | Env. | L3 | L2 | L1 | L0 |
| **R10** | Channel C amplitude | — | — | — | Env. | L3 | L2 | L1 | L0 |
| **R11** | Envelope period (low) | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| **R12** | Envelope period (high) | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| **R13** | Envelope shape | — | — | — | — | — | B2 | B1 | B0 |

| INFO | **In R7, a bit at 0 enables the channel; at 1 it disables it. In R8–R10, if the Env. bit is active, amplitude is controlled by the envelope (R11–R13) instead of L3–L0.** |
|---|---|

## I/O Ports — Two ZonX-81 Compatible AY Chips

The SD81 Booster implements two physical AY chips, compatible with the standard ZonX-81 interface, using partial address decoding: only bits A1, A2, A3, A5 and A7 are checked; bits A0, A4 and A6 are don’t-care.

| Chip | Register select (latch) | Data write |
|---|---|---|
| Chip A (standard ZonX-81, A3=1) | $CFh / $DFh | $0Fh / $1Fh |
| Chip B (SD81 Booster extension, A3=0) | $C6h | $06h |

Bit A7 acts as the AY's BC1 line: during a write, A7=1 selects the register (address latch) and A7=0 writes the data into the already-selected register. PSG status read-back (BC1=1, BDIR=0) is implemented in the FPGA but not currently exposed via firmware/BASIC.

## VGM Player Opcodes

The interface VGM player only interprets opcodes referring to the AY chip. The rest are ignored without producing an error.

| Opcode | Parameters | Description |
|---|---|---|
| **61h** | nn nn | Wait n samples (little-endian, 0–65535; ≈ 1.49 s maximum). Long pauses are represented with several consecutive commands. |
| **62h** | — | Wait 735 samples (1/60 of a second). Equivalent to 61h DFh 02h. |
| **63h** | — | Wait 882 samples (1/50 of a second). Equivalent to 61h 72h 03h. |
| **A0h** | aa dd | Write value dd to AY register number aa. |

## SP0256-AL2 Allophone Table

Allophones are used with the MCU BINARY SAY command (16h) for precise phonetic synthesis. High-level text access (LOAD *SAY "text") does not require knowing these codes.

| Code | Allophone | Example | Code | Allophone | Example |
|---|---|---|---|---|---|
| **$00** | PA1 | pause 10 ms | **$20** | AW | out |
| **$01** | PA2 | pause 30 ms | **$21** | DD2 | do |
| **$02** | PA3 | pause 50 ms | **$22** | GG3 | wig |
| **$03** | PA4 | pause 100 ms | **$23** | VV | vest |
| **$04** | PA5 | pause 200 ms | **$24** | GG1 | got |
| **$05** | OY | boy | **$25** | SH | ship |
| **$06** | AY | sky | **$26** | ZH | azure |
| **$07** | EH | end | **$27** | RR2 | brain |
| **$08** | KK3 | comb | **$28** | FF | food |
| **$09** | PP | pow | **$29** | KK2 | sky |
| **$0A** | JH | dodge | **$2A** | KK1 | can't |
| **$0B** | NN1 | thin | **$2B** | ZZ | zoo |
| **$0C** | IH | sit | **$2C** | NG | anchor |
| **$0D** | TT2 | to | **$2D** | LL | lake |
| **$0E** | RR1 | rural | **$2E** | WW | wool |

| Code | Allophone | Example | Code | Allophone | Example |
|---|---|---|---|---|---|
| **$0F** | AX | succeed | **$2F** | XR | repair |
| **$10** | MM | milk | **$30** | WH | whig |
| **$11** | TT1 | part | **$31** | YY1 | yes (short) |
| **$12** | DH1 | they | **$32** | CH | church |
| **$13** | IY | see | **$33** | ER1 | fir (short) |
| **$14** | EY | beige | **$34** | ER2 | fir (long) |
| **$15** | DD1 | could | **$35** | OW | beau |
| **$16** | UW1 | too | **$36** | DH2 | they |
| **$17** | AO | aught | **$37** | SS | vest |
| **$18** | AA | hot | **$38** | NN2 | no |
| **$19** | YY2 | yes (long) | **$39** | HH2 | hoe |
| **$1A** | AE | hat | **$3A** | OR | store |
| **$1B** | HH1 | he | **$3B** | AR | alarm |
| **$1C** | BB1 | business (short) | **$3C** | YR | clear |
| **$1D** | TH | thin | **$3D** | GG2 | guest |
| **$1E** | UH | book | **$3E** | EL | saddle |
| **$1F** | UW2 | food | **$3F** | BB2 | business (long) |

# Appendix H — Hardware sprites

The SD81 Booster includes a sprite system handled entirely by hardware: 64 sprites of 8×8 pixels, each with its own transparency mask and its own colour per row (several colours within the same sprite), automatically composited over the background image at no CPU cost. They are controlled with POKEs to a fixed address block, or more conveniently with four dedicated LOAD * commands.

## Coordinate system

A sprite’s position is expressed in a coordinate system offset by 32 pixels from the screen, so a sprite can enter and leave through any edge with a clean clip instead of abruptly reappearing on the opposite side:

sprite coordinate 32 = screen pixel 0

X: 0-318 (32 off-screen left, 256 visible, 32 off-screen right)

Y: 0-255 (32 off-screen above, 192 visible, 32 off-screen below)

Any sprite pixel that falls outside the real visible screen area (0,0)-(255,191) simply isn’t drawn. That’s why X needs 9 bits (two POKEs: low byte and high byte) while Y fits in a single byte.

## Address map (POKE 2100-2128)

Just like HFILE or the border pattern, these addresses fall within the first 8 KB of the Z80 memory map (ROM area): the interface intercepts these specific writes instead of letting them fall through, so no real memory is needed there. Before writing a sprite's colour, pixel data or mask, it must be selected first with POKE 2100,n.

| Address | Value | Function |
|---|---|---|
| 2100 | 0-63 | Selects the active sprite. Subsequent writes affect this sprite. |
| 2101 | 0/1 | Enables (1) or disables (0) the selected sprite. |
| 2102 | 0-255 | X coordinate, low 8 bits. |
| 2103 | 0/1 | X coordinate, bit 8 (full range 0-318). |
| 2104 | 0-255 | Y coordinate (32 = first visible line). |
| 2105-2112 | byte | Colour of each of the 8 rows: ink×16+paper (0-15 each). |
| 2113-2120 | byte | Pixel pattern of each of the 8 rows. |
| 2121-2128 | byte | Transparency mask of each of the 8 rows. |

## How each pixel is drawn

Three bits combine for each sprite pixel, one from each array at the same row and column:

**1. Mask —** if the bit is 0, that point is transparent and whatever is underneath shows through (background, text, another lower-priority sprite). If it's 1, the sprite draws something there.

**2. Pixel data —** only matters if the mask is 1: bit=1 uses that row's ink colour, bit=0 uses that row's paper colour.

Since colour is defined row by row (not a single colour per sprite), one sprite can mix several colours — for example, a ball with different coloured bands on each line.

## Colour format

Each row's colour byte is ink×16+paper, with ink and paper each on the 0-15 scale (the same one used by LOAD *COLOR and LOAD *BORDER): bit 3 = bright, bit 2 = green, bit 1 = red, bit 0 = blue. This format is the same in both CHROMA and SPECTRUM mode — the sprite's colour is applied after the hardware has already resolved which of the two modes is active, so there's no need to worry about the mode when defining a sprite's colour.

## Priority between overlapping sprites

If more than one sprite is active at the same pixel, the highest-numbered one wins (sprite 63 covers sprite 0). There is no other drawing-order control besides the sprite number.

## BASIC commands

**LOAD *SPRITE <n>,<x>,<y>** — selects sprite <n> (0-63), sets its position and enables it, in a single step.

LOAD *SPRITE 0,150,88

**LOAD *SPRITE <n> STOP** — hides sprite <n> without touching its position or graphic data.

LOAD *SPRITE 0 STOP

**LOAD *SPRCOL / LOAD *SPRPIX / LOAD *SPRMASK <n>,“<16 hex chars>”** — load, respectively, the colour, pixel data or mask of sprite <n>: 16 hexadecimal characters = 8 bytes = one row per byte. Each command automatically selects the given sprite.

LOAD *SPRCOL 0,“0202020606060606”

LOAD *SPRPIX 0,“070810182626181C”

LOAD *SPRMASK 0,“070F1F1F3F3F1F1F”

16 hex characters are used instead of a single 48-character block (colour+pixel+mask together) precisely to reduce the risk of a mistake when typing such a long string by hand.

## Capacity

The current firmware supports 64 sprites, with free per-pixel positioning and colour per row, but at most 12 of them can appear on the same screen line. If more than 12 active sprites cross the same line, the 12 with the highest index are drawn on that line and the others do not appear; on the lines where they do not coincide they show normally. It is a limit similar to the NES’s (which had 8 per line), and programs that move many sprites together usually rotate which ones are left out in each frame so that they flicker instead of disappearing. Any active sprite that crosses the line counts, wherever it is horizontally, and an active sprite off the screen also takes a slot: it is best to disable it (LOAD *SPRITE n STOP). A change to a sprite’s data shows from the next line. It is a fixed figure for the end user ---it depends on the specific FPGA bitstream installed in the interface---, not something configurable from BASIC.

# Appendix I — CP/M: an alternative operating system

Besides the ZX81’s built-in BASIC, the SD81 Booster can boot CP/M, the standard operating system of professional 8-bit microcomputers from the late 70s and 80s. Two ports exist for this hardware:

**CP/M 2.2,** the classic version, with no memory banking: up to about 41 KB of TPA (transient program area) in MC45 mode.

**Banked CP/M 3 (CP/M Plus),** which uses the SD81 Booster’s memory paginator to split RAM into independent 8 KB system and user banks. This is the one documented in this appendix.

CP/M 3 adds, over 2.2: more TPA (the system and the user no longer share the same bank), a RAM disk, the real-time clock integrated into file timestamps, a 256-character colour console, and a network driver for BBS and WiFi communication systems.

The system follows the Digital Research (DRI) standard as far as the hardware allows: same CCP, BDOS and BIOS format, same .COM program conventions, same public BIOS call table. What’s specific to this port is the memory paging layer (the XIOS) and the device drivers — disk, keyboard, console, clock and network — written for the SD81 Booster hardware.

## CP/M 2.2: the two build modes

CP/M 2.2 is distributed in two variants, selected at build time, depending on whether MC45 mode is used (see chapter 11, “Memory management”, and in particular “Extending MC45 to blocks 6 and 7”):

| Mode | MC45 | TPA | Video RAM | Recommended for |
|---|---|---|---|---|
| 32 KB | No | ~24 KB | $8000 | General CP/M use |

| Mode | MC45 | TPA | Video RAM | Recommended for |
|---|---|---|---|---|
| 48 KB | Yes | ~41 KB | $C000 | Turbo Pascal and large programs |

The 32 KB mode works on any SD81 Booster with nothing special to enable. The 48 KB one needs MC45 turned on, and gives almost double the TPA — the difference between being able to compile with Turbo Pascal or not.

CP/M 3 needs ALL memory above 32 KB for its bank paging, not just blocks 4 and 5 — hence it uses the MC45 extension to blocks 6 and 7 described in chapter 11 (POKE 2062,170). CP/M 3's own boot process turns it on by itself, with nothing for the user to do.

## Installation and boot

The system is distributed as a single file, SYSTEM.BIN, containing the BIOS, BDOS and CCP already linked together. It loads from BASIC just like any other machine code block and starts with USR:

> LOAD FAST "SYSTEM.BIN" CODE 24576
>
> RAND USR 24576

Besides SYSTEM.BIN, the card needs the disk images: A.IMG through D.IMG (256 KB of directory, 2 MB of data each) at the root of the SD card. CP/M 3 images use their own format and are NOT compatible with CP/M 2.2's or BASIC's — they can't be mixed. A fifth drive, E:, is a RAM disk that needs no image: it's created empty on every boot and its contents are lost on power off or reset.

## Banked memory

The SD81 Booster pages memory into 8 blocks of 8 KB. CP/M 3 uses that capacity to keep two complete contexts: a system one (where the BIOS and BDOS live) and a user one (the TPA, where programs run). The topmost block ($E000-$FFFF) is common to both contexts — that's where everything that needs to be reachable no matter which one is active lives, including the public BIOS call table (see below).

A user program doesn't need to know any of this: the system itself handles the context switch on every BDOS or BIOS call. It only matters if you're writing code that accesses memory outside the TPA, or that needs to be faster than a normal BDOS call (see "Calling the BIOS directly").

## Disks: A: to D: (SD card) and E: (RAM)

A: to D: are the four drives backed by the SD card, in standard CP/M 3 format (2 KB allocation block, 256 directory entries). E: is a RAM disk of about 390 KB, fast but volatile — useful for compiles, temporary files, or anything that doesn't need to survive a reset.

MOUNT3.COM lets you hot-swap the image mounted on a drive without restarting the system — needed to swap virtual floppies during a long session.

## Keyboard

The ZX81 doesn't have a full ASCII keyboard: it's missing symbols, and only has one arrow key per SHIFT combination. The table below summarises how each character is reached. See "CP/M keyboard table" for the full list.

**Direct and SHIFT+key:** letters, digits and the arrows (WordStar-style: SHIFT+5/6/7/8 = left/down/up/right).

**ENTER+key:** the symbols the ZX81 doesn't have printed directly — including the ones needed to program in C or use a BBS (@ \ | ~ ` _ # % & !, among others).

**SHIFT+ENTER, then a key:** control mode. Gives ^A-^Z with letters, and NUL with the space bar.

**SHIFT+9 and SHIFT+0:** backspace (BS, $08) and delete (DEL, $7F) — both, because different remote software expects a different byte for the same function.

**SHIFT+.:** tab (TAB, $09).

## Real-time clock

CP/M 3 stamps the creation/modification date and time of every file using the board's RTC. The clock is set through the SD81 Booster's own RTC (for example from BASIC, or by syncing it via NTP if a WiFi module is fitted) — CP/M only reads whatever the RTC already has.

## Console: 256-character colour terminal

The local console understands a subset of ANSI/VT100 sequences, enough for BBS, full-screen editors and any program that draws in colour:

**Cursor movement:** ESC[row;colH (or f), ESC[nA/B/C/D (up/down/right/left), ESC[s/ESC[u (save/restore position).

**Clearing:** ESC[2J (screen), ESC[K (end of line).

**Colour:** ESC[...m (SGR) — see the SGR utility below for the list of supported codes.

**ESC[6n (cursor position request):** actually answered, with ESC[row;colR — needed because some programs (BBS included) measure the terminal size this way before starting.

256 characters (full CP437 set, not just printable ASCII).

## Calling the BIOS directly

Every BIOS function is reachable from a user program without going through the BDOS, via a public jump table of 3 bytes each, starting at address $FF00:

| Function | Index | Address |
|---|---|---|
| CONST | 2 | $FF06 |
| CONIN | 3 | $FF09 |
| CONOUT | 4 | $FF0C |
| AUXOUT | 6 | $FF12 |
| AUXIN | 7 | $FF15 |
| AUXIST | 18 | $FF36 |
| AUXOST | 19 | $FF39 |

(address = $FF00 + index × 3; the full list of 33 entries follows the standard CP/M 3 order).

It's the same technique WordStar and Turbo Pascal use to avoid paying the BDOS cost on frequent console operations, and it's meant for programs that need the best performance possible — for example, a communications terminal. The TERM program included with the system (see below) is a real example: it calls CONOUT this way, and that multiplies console speed several times over going through the BDOS.

## Communications

The system includes a network driver (NET) that exposes a raw byte stream to the Z80, as if it had a modem attached — the software itself decides what to do with that stream (for example, sending AT commands to dial a connection). The real WiFi bridge is handled by the board's ESP32 module; the Z80 knows nothing about sockets or addresses.

TERM.COM is the terminal included to use NET. Unlike a typical CP/M program, it does NOT go through the AUX: logical device or the BDOS: it talks to the network hardware directly, with the same direct BIOS call trick described above for screen output. The reason is performance — a generic terminal honouring DEVICE AUX: would be several times slower — and it means TERM is built specifically for NET, not as a substitute for a general-purpose communications terminal.

TERM keys (all with ENTER+key):

| Key | Function |
|---|---|
| **ENTER+0** | Exit |
| **ENTER+9** | Toggle local echo (see it as you type) |
| **ENTER+8** | Toggle file logging (with TERM file.BIN) |

Before a connection is open, whatever the Z80 writes over NET is interpreted as Hayes modem (AT) commands. Once connected, everything written goes straight to the socket — AT commands included, unless command mode is re-entered with "+++". Supported commands:

| Command | Effect |
|---|---|
| AT | Check. Replies OK. |
| ATDT host:port | Opens the socket (a space after "DT" is optional). Replies CONNECT or NO CARRIER. |
| ATDL | Redials the last host:port dialed, whether it succeeded or not. ERROR if nothing was ever dialed. |
| ATH / ATH0 | Hangs up. Replies OK. |
| ATO / ATO0 | Returns to data mode without hanging up. Replies CONNECT. |
| ATE0 | Turns off command echo in command mode. |
| ATE1 | Turns on command echo (the default). |
| ATZ | Resets to the initial state: hangs up and restores echo. Replies OK. |
| +++ | Escapes to command mode without hanging up, with a one-second silence before and after. Replies OK. |

Responses always end in CR/LF: OK, CONNECT, NO CARRIER, ERROR. If the other end closes the connection, TERM gets NO CARRIER and switches back to command mode on its own — there is no automatic reconnection of the session: to dial again, use ATDT or ATDL, just like on a real modem.

There is also a configuration web page (http://<wifi-module-ip>/telnet) that lets you set the destination and check connection status without AT commands — mainly meant for testing. The path real software uses is ATDT/ATDL.

## Included utilities

| Program | Function |
|---|---|
| **MOUNT3.COM** | Hot-swaps the image mounted on a drive |
| **TERM.COM** | Communications terminal (see above) |
| **CLS.COM** | Clears the screen and homes the cursor |
| **SGR.COM** | Changes the console’s ink/paper colour |
| **BORDER.COM** | Changes the border colour |

SGR parameters sends the ANSI sequence ESC[parameters m to the console, separated by ;:

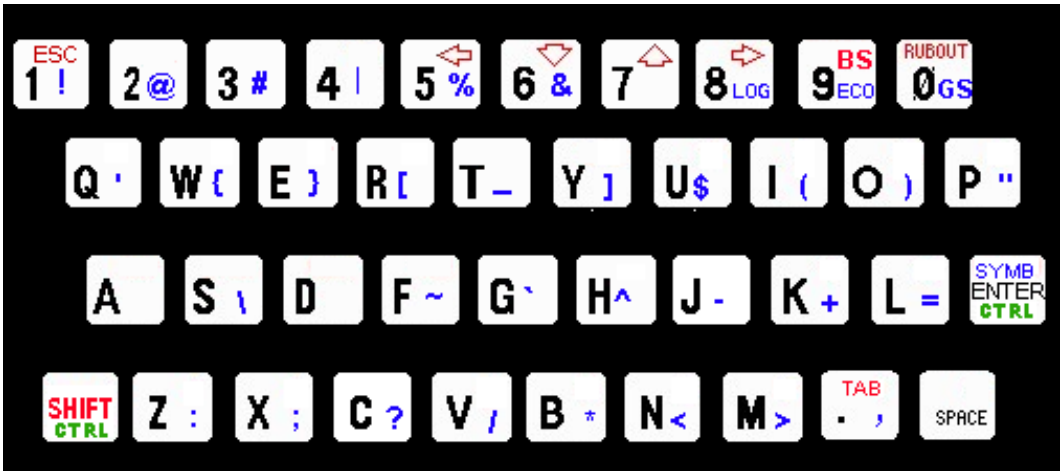
| Code | Effect |
|---|---|
| **0** | Restore default colours |
| **1** | Bright ink |
| **30-37** | Ink colour: black, red, green, yellow, blue, magenta, cyan, white |
| **40-47** | Paper colour, same colour table |

> SGR 0 reset
>
> SGR 33 yellow ink
>
> SGR 1;33;44 bright yellow ink, blue paper

BORDER n, with n from 0 to 7 (normal colour) or 8 to 15 (same colour, bright, by adding 8) — see Appendix E, “Chroma81 interface port (7FEFh)”, “Writing” section, for the full port format.

> BORDER 4 blue border
>
> BORDER 12 bright blue border

## CP/M keyboard table




ENTER+8 and ENTER+9 are labelled by their function (LOG, ECO) rather than by the control code they send ($1E, $1C): they’re the ones TERM uses for its own menu.

| Key | Direct | SHIFT | ENTER |
|---|---|---|---|
| **1** | 1 | ESC | ! |
| **2** | 2 | — | @ |
| **3** | 3 | — | # |
| **4** | 4 | — | \| |
| **5** | 5 | ← | % |
| **6** | 6 | ↓ | & |
| **7** | 7 | ↑ | — |
| **8** | 8 | → | — |
| **9** | 9 | BS | ^\ |
| **0** | 0 | DEL | ^] |
| **Q** | q | Q | ’ |
| **W** | w | W | { |
| **E** | e | E | } |
| **R** | r | R | [ |
| **T** | t | T | _ |
| **Y** | y | Y | ] |
| **U** | u | U | $ |
| **I** | i | I | ( |
| **O** | o | O | ) |
| **P** | p | P | ” |
| **A** | a | A | — |
| **S** | s | S | \ |
| **D** | d | D | — |
| **F** | f | F | ~ |
| **G** | g | G | ` |
| **H** | h | H | ^ |
| **J** | j | J | - |
| **K** | k | K | + |
| **L** | l | L | = |
| **Z** | z | Z | : |
| **X** | x | X | ; |
| **C** | c | C | ? |
| **V** | v | V | / |
| **B** | b | B | * |

| Key | Direct | SHIFT | ENTER |
|---|---|---|---|
| **N** | n | N | < |
| **M** | m | M | > |
| **.** | . | TAB | , |
| **SPACE** | space | space | space |
| **ENTER** | CR | CR | CR |

Also, SHIFT+ENTER then a letter gives ^A-^Z (letter AND $1F), and SHIFT+ENTER+space gives NUL.

## CP/M 3 memory map (summary)

| Range | Contents |
|---|---|
| **$0000-$DFFF** | Banked: TPA (user context) or BIOS/BDOS/CCP (system context), depending on which is active |
| **$E000-$FFFF** | Common to both contexts: the SCB ($FE00), the public BIOS call table ($FF00), and the rest of the state that must look the same from both sides |

The exact TPA size depends on the final system configuration; it can be checked by booting and looking at CP/M 3's startup message (Banked memory, NN.NK TPA).

## CP/M and the debugger

The hardware debugger (Appendix L) works with CP/M. Press the QuickSilva button for 1 to 3 seconds and its screen appears; S, O, U and C step or carry on, and pressing the button again also carries on. With CP/M, block 0 ($0000-$1FFF) is CP/M's RAM, so the sprites and the control POKEs, which live in that block, do not exist while CP/M is running. While the debugger is on screen, block 0 behaves again like the usual ROM and CP/M's RAM is left alone: when you carry on, everything is as it was.

Snapshots work too: the QuickSilva button for more than 3 seconds (or snap in the console) saves the CP/M state and LOAD *Z81 restores it. The history (th) keeps noting the instructions even though block 0 is RAM.

# Appendix J — The file explorer

The explorer (EXPLORER.BIN) is an SD card browser in Superfast HiRes Spectrum mode (42 columns, with a compressed 6-pixel font), meant to save having to remember file names or LOAD commands to load a program, view an image, or manage the card.

## Boot

```
2 LET ORG=24576
3 LOAD FAST
10 FAST
12 LOAD *FULLPAG
15 LOAD THEN CLEAR ORG-1
```

```
20 LOAD FAST “/EXPLORER/EXPLORER.BIN” CODE ORG

30 LET N=USR ORG

40 IF N=0 THEN GOTO 5000

50 LET F$=“”

55 LET EXT = 255

60 FOR I=0 TO N-1

65 LET A$=CHR$ (PEEK (ORG+3+I))

67 IF A$=“.” THEN LET EXT = I+1

70 LET F$=F$+A$

80 NEXT I

85 SLOW

90 LET E$=F$(EXT TO )

95 LOAD THEN CLEAR 32767

100 IF (E$ = “.P”) OR (E$=“.81”) OR (E$=“.WAV”) OR (E$=“.ROM”) THEN LOAD FAST F$ THEN
GOTO 1

130 IF (E$ = “.Z81”) THEN LOAD *Z81 F$

300 GOTO 10

5000 LOAD THEN CLEAR 32767

5010 SLOW
```

When you leave the explorer with a file selected (ENTER or 8 on a file that isn’t .VGM, .PEB, .SCR or .TXT — the explorer itself handles those four, see below), this stub collects the name in F$ and decides what to do based on the extension. As written, it loads .P, .81, .WAV and .ROM files with LOAD FAST and restores .Z81 snapshots with LOAD *Z81 (see 7.7); add your own condition before line 300 for any other extension you want to load differently.

Lines 95 and 5000 put RAMTOP back to 32768, the standard value for a 16K ZX81, which line 15 had lowered to ORG to protect the explorer. Note that it is 32767, not 32768: LOAD THEN CLEAR sets RAMTOP to the given address plus one, and some games hang if RAMTOP doesn’t hold the standard value.

The SD card image downloaded from the repository already ships configured to boot this way: it includes this startup as /AUTOEXEC.P, so with the interface plugged in, just power on the ZX81 and press RUN then ENTER for the explorer to load and start on its own, with nothing to type (see 5.4, “The AUTOEXEC program”).

## Navigation

| Key | Action |
|---|---|
| **6** | Move down the list |
| **7** | Move up the list |
| **1** | Previous page |
| **2** | Next page |

| Key | Action |
|---|---|
| **5** | Go up to the parent folder |
| **8 / ENTER** | Enter the selected folder, or open/load the file |
| **SPACE** | Exit to BASIC (with no file selected) |
| **.** | Filter the listing by wildcards (* ?) |
| **SHIFT+1** | Clear the active filter |

The filter persists while you navigate between folders; it’s only cleared with SHIFT+1.

## What happens when you select a file

Depending on the extension, ENTER/8 does one of these four things without leaving the explorer, or returns the name to BASIC:

| Extension | Behaviour |
|---|---|
| **.VGM** | Loaded and starts playing immediately |
| **.PEB** | Same as .VGM, for effects generated with the PEG editor |
| **.SCR** | Shown full-screen (Spectrum capture, 6912 bytes); any key returns to the listing |
| **.TXT** | Opened in the text viewer (see below) |
| **anything else** | The explorer exits to BASIC with the file name, for the stub to decide what to do |

## File operations

| Key | Action |
|---|---|
| **N** | New folder (asks for the name) |
| **D** | Delete the selected file or folder (asks for confirmation; folders are only deleted if empty) |
| **R** | Rename (asks for the new name, pre-filled with the current one) |
| **C** | Mark the selected item to copy |
| **X** | Mark the selected item to move |
| **V** | Paste (copies or moves, depending on what was marked with C/X, into the current folder) |

When you mark something with C or X, the matching icon in the bottom bar turns blue as a reminder that something is pending to be pasted.

## Text editing (new folder / rename / filter)

An editor with a real cursor (inserts and deletes at the cursor position, not just at the end):

| Key | Action |
|---|---|
| **letters, digits, .** | Literal character |
| **SHIFT+Z/X/C/V** | : ; ? / |
| **SHIFT+J/K/L** | - + = |
| **SHIFT+N/M/.** | < > , |
| **SHIFT+B** | * |
| **SHIFT+SPACE** | ` (backtick) |
| **SHIFT+5 / SHIFT+8** | Move the cursor left / right |
| **SHIFT+0** | Delete (the character to the left of the cursor) |
| **SHIFT+1** | Restore the original name (rename only) |
| **ENTER** | Confirm |
| **SPACE** | Cancel |

## Configuration panel (S)

The S key opens and closes a side panel with the FPGA and music player status, letting you change it without leaving BASIC:

| Key | Action |
|---|---|
| **W** | Toggle WRX (character generator in the 8-16K RAM) |
| **F** | Toggle full paging (FULLPAG) |
| **M** | Toggle MC45 |
| **A** | Toggle the character set between 64 and 128 |
| **J** | Edit the five joystick keys (up, down, left, right, fire) |
| **T** | Stop the current music (VGM/PEB) |
| **Y** | Pause it |
| **U** | Resume it |
| **I** | View /MAN/IP.TXT (the WiFi module's IP, if connected) |

The panel also shows, on its last line, the MCU, ROM and FPGA version (Mx.y,Rx.y,Fx.y). W/F/M/A/J/T/Y/U only act while the panel is visible.

## Text viewer (.TXT)

| Key | Action |
|---|---|
| **6 / 7** | Next / previous line |
| **1 / 2** | Previous / next page |
| **SPACE** | Return to the listing |

## Hex viewer

Opened from the configuration panel with the H key, on the file selected in the listing (not available for folders):

| Key | Action |
|---|---|
| **6 / 7** | Next / previous row |
| **1 / 2** | Previous / next page |
| **G** | Go to an address (hexadecimal) |
| **Z** | Toggle the right-hand column between ASCII and ZX81 character codes |
| **SPACE** | Return to the listing |

## How a file is returned to BASIC

The explorer lives in RAM as loose machine code (loaded with LOAD ... CODE), so it can't touch the BASIC program/variable area itself — it just returns the name of the selected file (for the types it doesn't handle internally) and lets the stub decide. On exit:

1. It copies the name (already in ASCII) to a fixed address, ORG+3 (24579 with the ORG above), which survives the video restore.

2. It restores standard video mode and the normal memory mapping (identity: each block to its own numbered page).

3. It returns the name's length in BC (0 if you exited with SPACE) — in ZX81, USR address as a function returns the contents of BC.

## Assembling

With pasmo (http://pasmo.speccy.org/):

```
pasmo explorer.asm EXPLORER.BIN
```

# Appendix K — The EightyOne-CrossPlatform emulator

The emulator, EightyOne-CrossPlatform, can be downloaded from https://codeberg.org/wilco2009/EightyOne-CrossPlatform

This appendix covers **how to use the SD81 Booster inside the EightyOne emulator**: enabling it, pointing it at a virtual SD card, and working with the emulator's windows, menus, snapshots, serial console and hardware debugger.

It does **not** describe the SD81 Booster's own commands (LOAD *..., PLAY, MAP, ...) or the hardware. For those, see the rest of this manual, which applies unchanged to the emulator.

## What is emulated

EightyOne emulates the complete SD81 Booster, not just its commands:

| Part | In the emulator |
|---|---|
| MCU (STM32) | Its protocol on ports $A7 / $AF, the file system commands, PLAY, VGM, PEG, speech, RTC, joystick |
| FPGA | 64 × 8 KB memory paging, MC45, Superfast and Spectrum video modes, text modes with wide screens, hardware sprites, double buffer, Chroma81, ZonX81 AY, simulated interrupts |
| Firmware debugger | The firmware's own DEBUGGER.cpp, running unchanged, plus the FPGA side of the hardware debugger |
| SD card | A folder on your computer |
| LED and QuickSilva button | Shown and operated from the Serial Console window |

Because the emulator runs the real firmware code for the debugger and the real SD81 ROM (SDBOOST.ROM) for everything else, programs and tools written for the real board behave the same way.

## Enabling the SD81 Booster

1. Open **Options → Hardware…** (F6).

2. Select the machine **ZX81**, **TS1000** or **TS1500**. The SD81 Booster is not available on the ZX80, Lambda, R470, TK85, Jupiter Ace or ZX97LE; its checkbox is greyed out there.

3. Tick **SD81 Booster**.

4. Press **SD Card…** and choose the folder that will act as the virtual card (see «The virtual SD card»).

5. Press **Apply / OK**. The machine restarts with the SD81 Booster fitted.

When the SD81 Booster is active, the emulator configures the rest of the machine to match the real board, and those settings are forced:

- Sound: **ZonX81 AY** chip.
- Joystick: **Programmable** (default keys: 5 left, 8 right, 7 up, 6 down, Space fire).
- High resolution: **WRX**.
- Colour: **Chroma81**, always on (there is no switch).
- Character generator: **QuickSilva**, enabled by the SD81 ROM when needed.
- The 8–16K area is RAM.

The SD81 Booster setting is stored in the INI file and remembered between sessions. The Save Configuration option (**Options → Configuration**) keeps it together with the other hardware settings.

## The virtual SD card

The "SD card" is a normal folder. Everything the real card would hold is in it:

> <SD folder>/
>
> AUTOEXEC.P run at start-up (if present)
>
> SYS/
>
> SDBOOST.ROM the SD81 ROM image (required)
>
> DEBUG.BIN the debugger's monitor (optional, see «The hardware debugger»)

> <key>.ROM alternative ROMs (see «Boot keys»)
>
> ... your programs, folders, .Z81 snapshots, VGM files, etc.

### Choosing the folder

- **Options → Hardware… → SD Card…** opens a folder chooser. On the first run it defaults to the SD81 folder next to the emulator.
- You can use any folder. Use the same naming rules as for the real card (see the rest of this manual).
- Changing the folder requires applying the hardware dialog, which restarts the machine.

### The ROM

The emulator reads `SYS/SDBOOST.ROM` from the virtual SD, exactly as the MCU does on the real board, so you can test a new ROM just by replacing that file and pressing **Hard Reset**.

If the file is missing, the emulator falls back to the ROM loaded by EightyOne's own ROM loader. Some functions (loading snapshots through the debugger monitor, for example) need the real SDBOOST.ROM.

### Editing the card while the emulator runs

The folder is read on demand, so you can add or change files with your file manager while the emulator runs. Directory listings (LOAD *DIR) always show the current contents. A file that the SD81 has open (for example a VGM that is playing) is held open by the emulator, like on the real card, and cannot be deleted from Windows until it is closed.

## Starting up: reset, boot keys and AUTOEXEC

| Menu | What it does |
|---|---|
| **Control → Hard Reset (Power Cycle)** (Shift+F2) | Reinitialises the machine and the SD81 Booster, as if powered off and on. The ROM is reloaded from the SD. |
| **Control → Soft Reset (RESET Line)** (F2) | Pulses the reset line. The SD81 Booster keeps its state and reloads nothing. |
| **Control → NMI** (Shift+F3) | Generates an NMI. |
| **Control → Pause** (F3) | Pauses the emulation. |

### Boot keys

The SD81 ROM looks at the number keys 0–9 during start-up to choose an alternative ROM (/SYS/<key>.ROM). In the emulator, **hold the number key while pressing Hard Reset** (or while the emulator starts). The emulator keeps the keyboard state across the reset so the ROM sees the key, as it does on the real machine.

### AUTOEXEC

If the SD root contains AUTOEXEC.P, the SD81 ROM runs it at start-up, as on the real board.

# Display, sound and joystick

## Display

- **Options → Display Settings…** (F5) and **View → Zoom / Border** work as for any machine. The SD81's extended screens (Spectrum 256 × 192, Superfast text, wide text modes) are shown in the normal window.
- **Options → TV Emulation…** (F10) adjusts brightness, contrast, colour, noise, ghosting and scanlines. The defaults suit the ZX81 + SD81 Booster combination.
- A reset returns the display to the ZX81 default and switches the Chroma colour off, like the real board.

## Sound

Sound from the AY chips (ZonX81, the FPGA's second AY and the MCU's AY used by PLAY, VGM and PEG) goes to the emulator's audio output. **Options → Sound…** (F8) and **Tools → Sound Output…** apply as usual.

## Joystick

- **Control → Connect Joystick 1/2** connects a physical game controller to the programmable joystick.
- The keyboard mapping can be changed with the JOY command (see the rest of this manual) or from the Hardware dialog.
- **Auto-fire** options are in the Control menu.

# Snapshots

Snapshots are .Z81 files. When the SD81 Booster is active they carry an extra [SD81BOOSTER] section with the board's state (see DOC/z81_snapshot_format.md for the format). A snapshot made with the SD81 restores the whole state: all 64 pages of memory, the page mapping, video mode, border, Chroma, wide screens, sprites, AY chips, a VGM that was playing, PEG programs, files that were open, and the current directory.

## From the emulator's menu

- **File → Save Snapshot…** (F12) saves a snapshot.
- **File → Load Snapshot…** (F11) loads one. Drag and drop onto the window only accepts .P files, not snapshots.

## From the ZX81

- LOAD *Z81 "name" loads a snapshot from the SD card. When the SD81's debugger monitor is installed (see «The hardware debugger»), the emulator loads it exactly like the real board does, so sound is silenced while it loads and the VGM, PEG and open files resume at the end.
- Snapshots taken from the hardware debugger (the snap command, or holding the QS button for 3 seconds) are normal .Z81 files and load from the emulator's menu too.

## Loading while the SD81 is not enabled

A snapshot with an [SD81BOOSTER] section loaded when the SD81 is disabled loads the memory and CPU only; enable the SD81 first to get the complete state.

## CP/M Disk Manager

**Tools → CP/M Disk Manager…** lets you browse CP/M disk images and copy files between them and your computer, without external tools.

- Supported formats: SD81 **CP/M+ (2 MB)** and SD81 **CP/M 2.2 (250 KB)**.
- You can create a new image, copy files to and from the image, and delete files.
- CP/M itself runs from the SD card (LOAD *CPM…); see the rest of this manual. **Help → CP/M Keyboard Map** shows the key mapping for CP/M.

## Debug Window: SD81 Booster panel

**Tools → Debug Window…** (Ctrl+F6) shows an extra **SD81 Booster** panel in its lower-right corner when the board is active. Values are greyed out while the emulator is running and become live when it is stopped.

| Field | Meaning |
|---|---|
| Mode | Current video mode (TEXT, Spectrum, wide, ...) |
| Chroma | Chroma81 mode register |
| HFILE | Display file address |
| Border | Border mode and colour |
| NMI gen | State of the NMI generator (and frames) |
| CDFLAG | FAST/SLOW flag state |
| WRX | WRX high resolution on/off |
| MC45 | MC45 mode: off (code up to $8000), on (whole range) or on ($C000) |
| Block / Page | The 8 memory blocks and the 8 KB page currently mapped to each |

The **Live Memory View** (Ctrl+F5) shows the memory as the CPU sees it, with the current mapping.

If the SD81 symbol feature of the Debug Window is used, symbol files (.SYM) are resolved per memory page, so labels in paged code resolve to the right page.

## The Serial Console

The real board has a USB-C serial port used for diagnostics and for the hardware debugger. The emulator provides the same thing in a window.

Open it with **Tools → SD81 Booster Serial Console…**.

### The window

> [ input line ] [QS (hold)] [LED] [Clear]
>
> +------------------------------------------------------------+
>
> | output of the MCU |
>
> | ... |

+----------------------------------------------------------+

- **Input line** (top): type a command and press **Enter**. The line is echoed in the output with a > prefix. The ↑ and ↓ keys recall previous lines.
- **Output**: everything the MCU prints, in a monospaced font. The window keeps the most recent part of a long output.
- **Clear**: empties the output.
- **QS (hold)** and **LED**: see below.

## The QS button and the LED

The board has a **QuickSilva button** and a three-colour **status LED**. In the emulator:

- Press and **hold the QS (hold) button with the mouse**, and release it when the LED shows the colour you want. Releasing performs the action for the time you held it.
- The **LED square** next to the button shows the board's LED colour in real time, including blinking.

| Held for | LED | On release |
|---|---|---|
| less than 1 s | green (or cyan when QuickSilva is off) | toggles QuickSilva |
| 1 – 3 s | magenta | pause / continue the program (hardware debugger) |
| more than 3 s | yellow | takes a snapshot |

Other LED colours reflect activity: yellow blinking while a snapshot is being written, cyan blinking while one is being loaded, magenta when the program is stopped by the debugger.

| INFO | **The LED and the held-time evaluation update once per emulated frame, so the colour can lag the mouse by up to one frame (20 ms). The button is not evaluated while the emulator itself is paused.** |
|---|---|

## Console commands

| Command | Action |
|---|---|
| DBG_PAUSE | Stops the program at the next instruction boundary (hardware debugger) |
| DBG_RELOAD | Reloads /SYS/DEBUG.BIN and resets the Z80 and FPGA |
| DBG_TRACE / DBG_TRACE on / DBG_TRACE off | Shows or changes the tracing of MCU commands in the console |
| DBG_FPGA | Dumps the emulated FPGA and mapper state (emulator only, for diagnostics) |
| Lowercase commands (r, s, m, ...) | The firmware debugger commands (see «The hardware debugger») |

Menu shortcuts: **Control → SD81 hardware debugger: pause** and **reload monitor and reset** send the corresponding DBG_ commands for you. Pressing **Up and Down** together on the joystick also pauses the program.

## Messages

The console also shows the MCU's status messages. With tracing on (DBG_TRACE on) you will also see a line for each command received from the Z81 (COMMAND=... and AFTER COMMAND=...) that is useful when reporting a problem.

# The hardware debugger

The SD81 Booster has a hardware debugger: the FPGA stops the program at an instruction boundary and calls a small monitor program (/SYS/DEBUG.BIN) kept in memory page 63, and the MCU provides the command line over the serial port. The emulator implements all of it, with the firmware's real debugger code, so it behaves like the real board. The commands, the debugger screen and the QS button are described in Appendix L; this section explains how they are operated in the emulator.

## Setting it up

1. Place the monitor file at **`SYS/DEBUG.BIN`** in the SD folder (8 KB at most). It is loaded at every hard reset, and the console prints Debug monitor loaded (/SYS/DEBUG.BIN, … bytes). If the file is missing, the debugger is off.

2. Make sure the SD ROM is a version that supports the debugger.

3. Open **Tools → SD81 Booster Serial Console…**.

## Using it

1. Pause the program with p in the console, the **Control → SD81 hardware debugger: pause** menu item, the QS button (1–3 s), or Up + Down on the joystick.

2. Type debugger commands in the input line. The firmware's own help lists them all; type ? to see it. In short:

| Group | Commands |
|---|---|
| Running | p pause, c continue, s [n] step, o step over, u step out, g addr run to address |
| Breakpoints and watchpoints | b addr, bc, bl; w r\|w\|io addr |
| Inspecting | r registers, d disassemble, m dump memory, io port |
| Changing | e addr bytes, x reg=val |
| Trace | t, th (history) |
| Symbols | sym file (pasmo .sym); addresses accept symbol or symbol+n |
| Screen | ui shows the debugger's own screen on the ZX81 (80 × 24 text); v shows the program's video while stopped |
| Snapshots | snap [-a] [name] writes a .Z81 file |

Numbers are hexadecimal. Programs running in **SLOW** and **FAST** mode are both supported.

## Typical session

> > p stop the running program
>
> > r show registers and the current instruction
>
> > b 8000 set a breakpoint
>
> > c continue; the program stops at 8000
>
> > s 5 step five instructions
>
> > snap save a snapshot of the stopped program
>
> > c carry on

### Snapshots from the debugger

snap writes a .Z81 file on the SD card (the name defaults to that of the last program loaded). Load it with LOAD *Z81, with File → Load Snapshot…, or on the real board. By default the pages assigned to some block and those the program has written since the last reset are saved; snap -a saves all 64.

## Command line and INI options

### Command line

On the command line and on drag and drop, the emulator only accepts .P files.

### INI options for automatic starts

Under [MAIN] in the INI file:

| Key | Meaning |
|---|---|
| AutoType | A sequence of **keys** typed automatically once the machine has booted, as if typed at the keyboard. \n is ENTER. Example: AutoType=J*CD”MYPROJECT”\nR\n types LOAD *CD”MYPROJECT” then RUN, which runs AUTOEXEC.P from that folder. |

Keys, not text: on a ZX81 LOAD is the single key J; symbols that need SHIFT are handled by the emulator. This is meant for quick start-up and testing, not for normal use.

The SD folder is stored in the INI with the key SD81VirtualSD, and the SD81 on/off state with the key SD81.

## Differences from the real hardware

The emulator reproduces the board, but some things are different by nature:

- **The SD card is a folder.** There are no FAT-format limits, the card is never “removed”, and file names follow the host file system (Windows is case-insensitive).
- **Timing.** The MCU and the SD are instantaneous compared with the real thing; programs that depend on SD speed run faster. VGM, PLAY and PEG timing follow the host clock.
- **Firmware and FPGA update commands** (LOAD *FPGA, firmware update) have no effect in the emulator.
- **Hardware not present:** Wi-Fi and the file server, the real-time clock battery (BAT) and the board’s LEDs other than the status LED.
- **Serial Console** replaces the USB-C serial terminal; there is no baud rate or port to configure.
- The status LED and the QS button are operated with the mouse in the console window; the real board’s physical button feel (short vs long press) is reproduced by how long you hold the mouse button.

## Troubleshooting

| Problem | What to check |
|---|---|
| The SD81 Booster checkbox is greyed out | The selected machine does not support it. Choose ZX81, TS1000 or TS1500. |
| Black screen / nothing runs after enabling it | SYS/SDBOOST.ROM is missing or not valid in the SD folder. Check the SD path in **Hardware → SD Card…**. |

| Problem | What to check |
|---|---|
| LOAD *Z81 shows a black screen at the end | Use a recent SDBOOST.ROM; with an old ROM the snapshot cannot be loaded through the debugger monitor. Loading from **File → Load Snapshot…** always works. |
| The console says “No debug monitor loaded” | SYS/DEBUG.BIN is missing or larger than 8 KB. Add it and use **Hard Reset** (or DBG_RELOAD). |
| The console shows nothing | Press **Clear** and send a command; check that **Tools → SD81 Booster Serial Console…** is the window you are typing in. |
| A file cannot be deleted from Windows | The emulator holds it open (for example a playing VGM). Stop the playback from the program or reset the machine. |
| The display is wrong after switching modes | Do a **Hard Reset**. If you can reproduce it, save a snapshot and report it. |
| Sound from the AY chips is missing | Check **Options → Sound…**, that sound output is enabled, and that the SD81 is active (it forces the ZonX81 AY). |

If you need to report a problem, turn tracing on with DBG_TRACE on, reproduce it, and copy the console output.

# Appendix L — Hardware debugger

The SD81 Booster includes a hardware debugger: it can stop any program at any time, show its registers and memory, run it step by step, set breakpoints and watchpoints, save the whole machine state to a snapshot and load it back. The program does not need to be prepared for it.

## What you need

The debugger is a small machine code monitor, /SYS/DEBUG.BIN, which the interface loads at power-up into an extended RAM page reserved for it (page 63). If the file is not on the SD card the debugger is not available and everything else works as usual. The MCU firmware, the ROM and the FPGA must come from the same package version.

It can be used in four ways: with the QuickSilva button and the ZX81 keyboard, with nothing else; from the USB console (see 15.4); from a web browser, with the WiFi module; and from the program itself, with the OUT trap.

## The QuickSilva button

| Press | What it does |
|---|---|
| **Short (under 1 s)** | Toggles QuickSilva, as always. |
| **1 to 3 s (the LED turns magenta)** | Pause: the program stops and the debugger screen appears. With the program stopped, another 1 to 3 s press resumes it. |
| **Over 3 s (the LED turns yellow)** | Snapshot: saves the state to a .Z81 file and the program carries on. |

While the program is stopped the STAT LED is magenta. A program in SLOW mode stops when it reaches its next video interrupt.

## The debugger screen

When paused with the button, an 80-column screen appears on the ZX81 itself with the registers, the flags, the breakpoints, the disassembly around the PC (highlighted), the stack and a memory dump. It is driven with the ZX81 keyboard and each key acts when released. Addresses are typed in hex: with 4 digits they are taken at once, with fewer press ENTER, and any other key cancels.

| Key | Action |
|---|---|
| **S** | One step: runs one instruction. |
| **O** | Step over: CALL, RST, DJNZ, LDIR and the like run in full. |
| **U** | Step out of the current routine (to the return address on the stack). |
| **C / SPACE** | Continue. |
| **G address** | Run to that address. |
| **B address** | Set or clear a breakpoint (B ENTER: at the PC). |
| **W R/W/I address** | Read, write or I/O watchpoint: stops right after the access (W ENTER clears it). |
| **R register value** | Change a register: P (PC), S (SP), A (AF), B (BC), D (DE), H (HL), X (IX), Y (IY), I. |
| **E address bytes ENTER** | Write bytes to memory (2 digits each). |
| **D address** | Disassemble from that address (D ENTER: back to the PC). |
| **M address** | Memory dump from that address. 5, 6, 7 and 8 move it. |
| **T number** | Trace: runs that many instructions one by one (T ENTER: up to a breakpoint). SPACE stops it. |
| **H** | Swaps the disassembly for the last instructions run (see Trace and history). |
| **V** | Show the program’s screen (V goes back to the debugger). |
| **Z** | Snapshot, staying stopped. |
| **L** | Load the last snapshot saved in this session. |
| **Q** | Hide the debugger screen (the program’s screen stays). |

On the program’s screen (after V or Q) the keyboard is still active: S saves a snapshot and carries on, Z saves a snapshot, L loads the last one, C or SPACE continue and V goes back to the debugger. Once opened, the debugger screen comes up at every stop (breakpoint, step...) until it is turned off from the console with ui.

## Snapshots

A snapshot saves the whole machine state to the SD card, in an extended EightyOne .Z81 format: memory, registers, the extended RAM pages and their assignment, the control POKEs, the Chroma81 colours, the sprites, the sound chips and the interface state (current directory, open files, VGM music...).

It is saved in the current folder with the name of the last program loaded plus a number (MAZOGS001.Z81, MAZOGS002.Z81...), or NONAME001.Z81 if nothing has been loaded. Save it with the QS button (over 3 s), with Z on the debugger screen (or S on the program’s screen, which also carries on) or from the console:

> snap [-a] [NAME]

By default the extended RAM pages assigned to some block are saved, plus those the program has written since the last reset (page 63, the debugger's, never). A LOAD does not clear that record, so a program that loads its data from the SD card in several goes keeps everything. With -a all 64 pages are saved. To load it back, use LOAD *Z81 "NAME.Z81" (see 7.7), the file explorer or L on the debugger screen.

## The USB console

With the USB-C port connected to a PC and a serial terminal open (see 15.4), everything above is available as text commands; h lists them all. Numbers are hex and an address can also be a symbol. The main ones:

| Command | What it does |
|---|---|
| **p / c** | Pause / continue. |
| **s [n]** | n steps (1). |
| **o / u** | Step over / step out. |
| **g addr** | Run to addr. |
| **b addr / bc [addr] / bl** | Set / clear one or all / list breakpoints. |
| **w r\|w\|io addr / w** | Set / clear the watchpoint. |
| **r / x reg=value** | Show the registers / change one. |
| **d [addr] [n] / m addr [n]** | Disassemble / memory dump. |
| **e addr bytes / io port [value]** | Write to memory / read or write a port. |
| **snap [-a] [name]** | Snapshot. |
| **t [n] / th [n]** | Slow trace / show the last instructions. |
| **tron / troff** | History on / off. |
| **sym [file \| ! \| -]** | Load / reload / clear the symbols. |
| **ui** | Debugger screen on the ZX81 on / off. |
| **v / v addr** | With the program stopped, show its screen in Superfast text / its WRX bitmap at addr. |

## From a web browser

With the WiFi module (see 9.7), the page http://IP-address/debug, linked from the file server as Debugger, shows the debugger in panels: disassembly (click the margin to set or clear a breakpoint, double click a line to run to it), registers and flags (click to change a register), breakpoints and watchpoint, stack, memory dump (click a byte to change it) and a console with the same commands as the USB one. The MCU and the WiFi module firmware must be updated together. Below the panels, Screen capture (the Capture button) draws in the browser the image the ZX81 is showing: it stops the program for a moment (of the order of tenths of a second, depending on the mode), reads what it needs from the video memory and lets it carry on. It is a photo, not video: the link speed does not allow more. It draws native ZX81, Superfast text of 32, 70 and 80 columns, HiRes and Spectrum, with the Chroma81 colour, the fine scroll and the sprites (with their limit of 12 per line, except in 70 and 80 columns, where sprites are not implemented). If the program is already stopped, the capture does not make it carry on. Modes it does not recognise, such as WRX with the font in I, are not drawn. The border is the Chroma81 one; in Spectrum mode it is not known and is drawn the same. The cap console command makes the capture without the web (only for testing). If the program uses an alternative D_FILE (POKE 2096-2098), the capture also brings the system one and the alternative D_FILE checkbox chooses which one is drawn with an older FPGA: in that case the interface cannot know whether the FPGA has the alternative one active (a POKE 2045,85 switches it off and going back to 170 does not switch it on), so if the image comes out mixed up, try the

other option. The sprites checkbox shows or hides them; unchecked before capturing, they are not read and the capture is faster. The capture's information line says whether the video state (mode, alternative D_FILE and attributes, double buffer) comes from the FPGA, which reports it from the version of this package, or was deduced from the POKEs; if it comes from the FPGA the checkbox is disabled, since there is nothing to choose. Below the capture there is a ZX81 keyboard: click its keys (SHIFT stays down until the next key) or tick "use my keyboard" so that the keys of your computer go to the ZX81. Letters, digits, ENTER, the full stop and the space bar go straight through; symbols are sent with the SHIFT they need on the ZX81 (for example, the quote is SHIFT+P), Backspace is RUBOUT (SHIFT+0), the arrows are SHIFT+5 to SHIFT+8 and Escape is BREAK (SHIFT+SPACE). The keys reach the ZX81 as if they were really pressed, also for the debugger screen. If the page stops sending (it is closed, the connection is lost), the interface releases all the keys after 1.5 seconds. The FPGA, the MCU firmware and the WiFi module firmware must be updated together.

Open the page with the module's IP address (LOAD THEN PRINT "*IP") rather than with the name sd81booster.local: on some computers the browser takes 10 to 30 seconds to resolve the name on every operation and the commands seem slow.

Commands that need the program stopped (breakpoints, registers, memory, steps) warn with a red line at the top if the program is running: press Pause first. In the page's console, History is th 20, Trace 100 runs 100 instructions in slow trace and Symbols reloads the symbol file (see Symbols).

## Symbols

If next to the program there is a symbol file with the same name and the .SYM extension (MYGAME.P and MYGAME.SYM), it is read when the program is loaded: the disassembly shows the labels and addresses can be typed by name (b LOOP, g START+10). It is the file the pasmo assembler writes when given a third argument:

```
pasmo mygame.asm mygame.bin MYGAME.SYM
```

It can also be loaded by hand with sym FILE.SYM. The symbols are read when the program is loaded and are not read again by themselves: if you reassemble and change the .SYM without loading the program again, the debugger keeps the old one and the labels show at addresses that are no longer the program's. sym ! reloads it, and sym alone says how many symbols there are and from which file. In the web page the Symbols button does the same and, with a name typed in the box next to it, loads that file.

## Trace and history

The interface keeps noting the address of every instruction the program runs, without slowing it down: when it stops, for whatever reason, th 20 in the console (or H on the debugger screen) shows the last instructions that led there. troff and tron turn it off and on.

The slow trace (t n in the console, or T on the screen) runs n instructions one by one and also notes the registers before each one, which th shows. It runs at about 200 instructions per second.

## For programmers: the OUT trap

A program can stop itself at a given point by writing 16 to port $3FEF; if the debugger is not loaded, nothing happens:

```
LD BC,$3FEF

LD A,16

OUT (C),A
```

## Limitations

- Extended RAM page 63 and port $3FEF are reserved for the debugger.
- A breakpoint in code that is loaded later (or that the program overwrites) is lost. For those cases, use G or a write watchpoint.
- If the program stops in the middle of an SD operation (a LOAD, for instance), let it carry on before using other debugger functions.
- While stopped, a SLOW or FAST program has no picture of its own: V shows it in Superfast text. Programs that make their own picture (WRX, pseudo hi-res) may not look right.
- With CP/M running (block 0 in RAM) the sprites and the control POKEs are off; the rest of the debugger works the same (see Appendix I).

# Appendix M — Networking for programmers: MQTT and turn-based games

The WiFi module (9.7) includes an MQTT client, cryptographic hashing and random numbers, all through AT commands. You type them in the TELNET terminal (9.8) or a program of your own sends them through the interface's network bridge (the MCU's NET_READ and NET_WRITE commands, see section 15). On top of that, the Z80 has a library of rooms and signed messages (net.inc) and a controller for turn-based games (game.inc), which are what the games in 9.9 use. The sources are in EXAMPLES/NETCHAT and EXAMPLES/NETGAME in the repository.

## MQTT commands

| Command | What it does |
|---|---|
| **AT+MQTT=host,port[,tls[,user,password]]** | Connects to that server. tls is 0 or 1. Blocks for up to about 8 seconds if the server does not answer. |
| **AT+MQTT** | With no arguments: tries the servers in /SYS/NET.CFG in order, remembers the last one that worked and moves to the next if one fails three times in a row. |
| **AT+MQTTID=string** | Sets the client identifier (by default, sd81- and 8 random hexadecimal characters). |
| **AT+MQTTWILL=topic,retain,text** | Last-will message: published if the connection drops without a goodbye. Send it before connecting. |
| **AT+MQTTSUB=topic[,qos]** | Subscribes to a topic (wildcards + and # allowed). Waits up to 3 seconds for the confirmation. |
| **AT+MQTTUNSUB=topic** | Cancels the subscription. |
| **AT+MQTTPUB=topic,qos,retain,len** | Publishes a message: the modem answers > and then len bytes are sent as they are (any value, up to 120). With len 0 it publishes an empty message, which clears the retained one. With qos 1 it waits 3 seconds for the confirmation and retries once. |
| **AT+MQTTRX** | Takes a message from the queue: +MQTTRX:topic,len, a line break and the len bytes; or just OK if there is none. |

| Command | What it does |
|---|---|
| **AT+MQTT?** | Status: +MQTT:state,server,queued,lost,error. |
| **AT+MQTTCLOSE** | Closes the MQTT session. ATZ closes the MQTT session and the TCP socket and forgets everything. |

- Data goes by length, in binary: the modem does not interpret the bytes that follow the >.
- Reception is by polling (AT+MQTTRX), never with messages that appear by themselves. The module keeps up to 8 messages of 128 bytes; when full it drops the oldest and counts it in lost. A message over 120 bytes or with a topic over 64 is dropped.
- There is a single MQTT session, independent of the TCP socket of ATDT. It reconnects by itself, waiting 2, 4, 8, 16 and 30 seconds, and subscribes again to whatever it was subscribed to.

## Cryptographic hashing and random numbers

| Command | What it does |
|---|---|
| **AT+SHA256=len** | The modem answers > and len bytes are sent. Returns +SHA256: and 64 hexadecimal characters. |
| **AT+HMAC=keylen,datalen** | After the >, the key and the data one after the other. Returns +HMAC: and 64 hexadecimal characters (HMAC-SHA256). |
| **AT+RAND=n** | Returns +RAND: and n random bytes in hexadecimal (n up to 32), from the module's generator. |

## The servers: /SYS/NET.CFG

A text file in the SYS folder of the SD card, with one server per line (up to 6): broker, the name, the port and, if TLS is needed, the word tls. Blank lines, lines that start with # and malformed lines are ignored. Without the file (or without any valid line) these three are used, without TLS:

> broker test.mosquitto.org 1883
>
> broker broker.emqx.io 1883
>
> broker broker.hivemq.com 1883

They are public test servers, with no guarantee of service. AT+MQTT with arguments ignores the file.

## States and errors

| Value | Meaning |
|---|---|
| **state 0 / 1 / 2 / 3** | off / connecting / connected / reconnecting |
| **error 0** | none (it is the last error: it clears when a connection succeeds) |
| **error 1** | no WiFi |
| **error 2** | cannot open the TCP socket |
| **error 3** | the server rejects the connection |
| **error 4** | timeout |

| Value | Meaning |
|---|---|
| **error 5** | bad arguments |
| **error 6** | queue or buffer full |
| **error 7** | no MQTT session |
| **error 8** | no confirmation for a qos 1 message |

## Rooms and messages in the games

Anyone can read the topics of a public server, so the room code is not in the topic. The code is 10 characters from an alphabet of 32 (50 bits), taken from AT+RAND. The topic is sd81/ followed by the first 12 hexadecimal digits of SHA256("topic"+code), and the key is SHA256("key"+code) in hexadecimal, used as it is as the HMAC key. Whoever sees the topic does not get the code.

Each player publishes, retained and with qos 1, to <topic>/0 (whoever created the room) or <topic>/1 (whoever joined). The message is: version (1 byte), sequence (2, starting at a random value and going up per sender), turn (2), data (0 to 100) and the first 8 bytes of the HMAC of everything before. The receiver drops what does not verify, what comes from its own slot and a repeated sequence.

Presence goes in <topic>/p0 and <topic>/p1, retained: 1 on entering and 0 on leaving; the last-will message publishes the 0 if the connection drops without a goodbye (the server takes about 90 seconds to notice). Before entering, the library reads both presences: that is how it knows whether the room is full, does not exist or a place has to be taken again.

This protects against pranks, not against a determined attacker: 50 bits can be broken offline, and the data travels in the clear.

## The net.inc library and the game.inc controller

net.inc (EXAMPLES/NETCHAT) is included in the program and needs about 1.2 KB of working RAM, whose address is defined with NB_BASE. Routines: net_init, room_new, room_join, room_send, room_poll, room_leave and, for a game loop, room_poll_send and room_poll_recv (the poll in two steps, which lets you draw in between). They return a code in A: 0 fine, 1 the modem said ERROR, 2 timeout, 3 unexpected answer, 4 invalid code, 5 message too long, 6 room full, 7 the room does not exist, 8 the room existed but is empty (a slot has to be chosen with room_join_as).

game.inc (EXAMPLES/NETGAME) is the controller of a turn-based game: the move, the turn, the state hash, the full state, the rematch and the reconnection. It knows nothing about the rules: the game defines six routines, G_STATE_LEN (the size of the state) and G_MOVE_LEN (1 or 2 bytes per move). Player 1 moves on an even move number and player 2 on an odd one.

| Routine | What it does |
|---|---|
| **g_reset** | Sets up the starting position. |
| **g_valid** | HL = the move, B = the player: returns CF = 1 if it is not legal. |
| **g_apply** | HL = the move, B = the player: makes it. |
| **g_winner** | A = 0 goes on, 1 or 2 that player wins, 3 draw. |
| **g_save / g_load** | HL = destination / source: the G_STATE_LEN bytes of the state. |

The controller’s messages are: 1 move (with the hash of the resulting state), 2 I ask for the state, 3 full state and 4 rematch; any mismatch is fixed by asking for the full state. The source of connect4.asm is the shortest example and chess.asm the most complete one (HiRes, clocks).

### Trying it by hand

With the TELNET terminal (9.8), type these lines on two interfaces, or on just one and publish with a desktop MQTT client (for example mosquitto_pub):

> AT+MQTT=test.mosquitto.org,1883
>
> AT+MQTTSUB=sd81test/hello,1
>
> AT+MQTTPUB=sd81test/hello,1,0,5
>
> >hello
>
> AT+MQTTRX
>
> AT+MQTT?

User Manual SD81 Booster v1.0 — Open hardware and open source software

# Review sources

- MANUAL/EN/SD81_Manual_EN.md
- release_notes_v1.7.0.md
- Arduino/bootloader/bootloader.ino
- Arduino/SD81BoosterV2_039_STM32/SD81BoosterV2_039_STM32.ino
- Arduino/SD81BoosterV2_039_STM32/WIFI_HANDLER.cpp
- Arduino/Wifi_module_01/Wifi_module_01.ino
- TOOLS/build_release.py