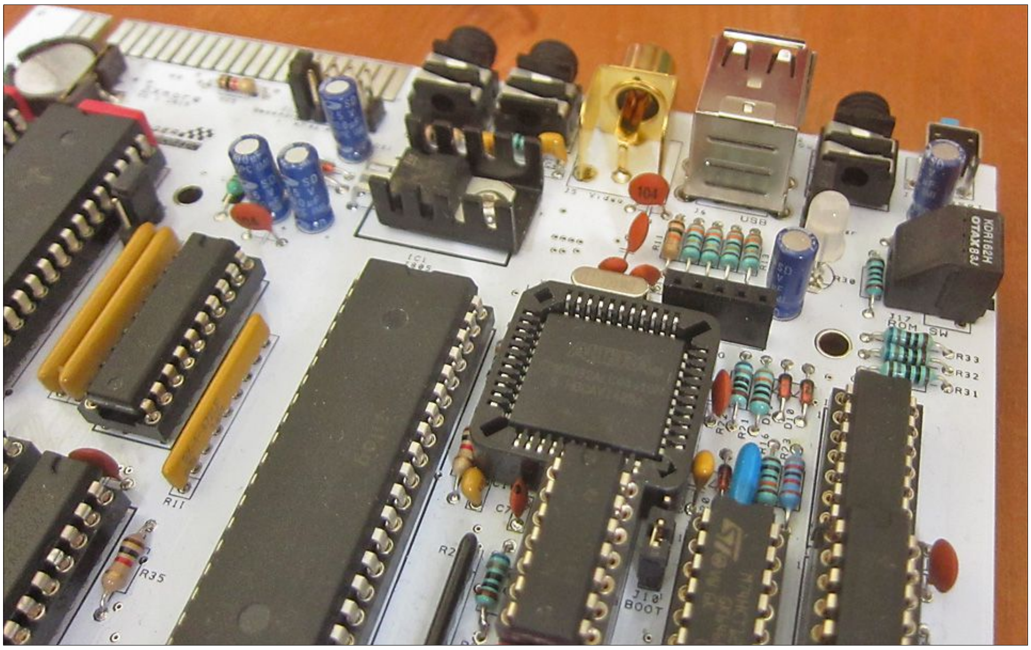
ZXmore & ZXmaster

User manual



designed by ginger-electronic.com

Contents

Description:

This manual describes the operation of ZXmore with the supplied firmware ZXmaster.

ZXmore is an 8 bit computer system which is compatible to many systems based on the Z80 processor, especially to the Sinclair systems ZX80, ZX81 and ZX Spectrum plus the CP/M system developed Digital Research (*). Programs and data can be stored on and read from a simple USB flash medium (USB drive).

It is possible to use a serial terminal via an optional USB/RS232 adapter or just use the inbuild keyboard and connected monitor or TV to the video connector in 40 char mode. An USB flash medium will also be used as disk drive to store and read data via CP/M.

The ZXmore is equipped with an inbuild keyboard with 40 keys, a video (RCA) connector for use with monitors or a TV (composite video monochrome) and 2 USB host ports for connection of USB mass storage devices and other peripherals. Original hardware modules for ZX80 or ZX81 can be used via the build-in expansion port (slot edge connector).

The ZXmore has 512k flash ROM and 512k RAM which is divided into eight separate instances with 64k ROM and 64k RAM which overlaps in the 64k address room. The border between ROM and RAM can be moved in steps of 4k in any direction. The first instance (0) maintains all other instances and deals with additional hardware like USB peripherals, video output and keyboard input.

The other seven instances can be configured with different options and different operating systems (firmware ROMs) which can be operated in parallel changed by keypress or even in multitasking mode. Several systems or firmware ROMs can be used concurrently without starting them new when they are switched. The memory layout can be configured individually per instance and any additional driver can be loaded during startup.

The firmware ZXmaster handles the configuration and operation of all instances or operating systems. The available ROM images to be used with the ZXmore are published under GPL or CC (creative commons) and may be used freely by the user. These additional firmware ROMs are not part of ZXmaster and ZXmaster is not based on these ROMs. ZXmaster can clone any Z80 system and is not fixed on some special ROM as long as hardware drivers for dealing with video display and keyboard are available or adapted.

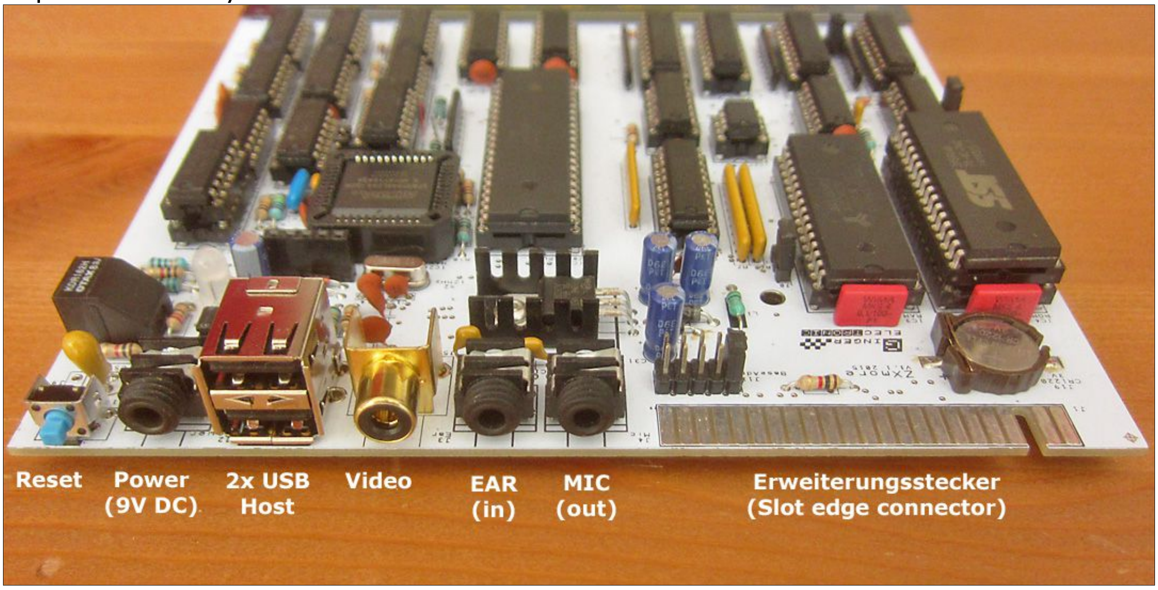
More technical information on ZXmore, ZXmaster and available firmware ROMs to use or supported are available in the appendix.

The manual refers to the first release 0.9 of ZXmaster with basic functionality which will be improved with newer versions from time to time.

(*) The current available release 0.9 of ZXmaster supports only programs for the systems ZX81 and ZX80 and loading programs from a USB flash medium. Further releases support saving program and data on USB as well as support of ZX Spectrum with monochrome display and use of CP/M.

Connectors:

In the picture below you will find all connectors of ZXmore:



Reset allows reset of the current instance and return to the ZXmaster control software.

Power is a 3.5mm audio jack for connection of a power supply with 9V DC, 250mA. Polarity is plus at the tip and minus at the ring of the plug. Alternatively the ZXmore can be supplied with power with a USB power supply (5V).

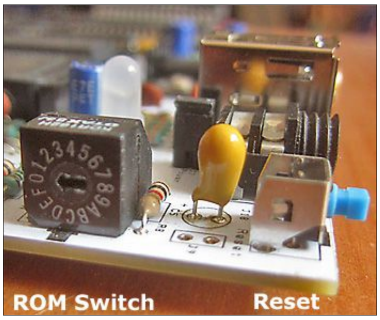
USB represents two host ports and can be used to connect flash drives or other peripherals

Video is a RCA jack for connecting a monitor or TV as display with corresponding input. The signal is of type composite video monochrome.

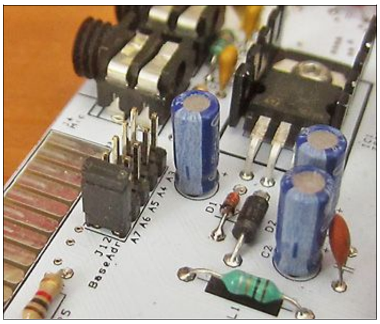
EAR is used to connect an audio cassette recorder to load a program from an original ZX80 or ZX81 cassette (for compatibility).

MIC is used to save programs or data to an audio cassette with audio signals instead of USB.

Slot edge connector is the expansion port for using ZX80 oder ZX81 hardware add-on's.



ROM Switch can be used to switch the instances manually in position 1-7 or by control of the ZXmaster software in position 0. Positions 1-7 can be used with ZX80 or ZX81 only (corresponding firmware ROM programmed in flash ROM assumed) due to a suitable default configuration.
ZXmaster can configure the ZXmore for some more firmware ROMs like for ZX Spectrum or CP/M.



J12 can be used to choose a base i/o address for the additional hardware features provided. Default address to be used should be 0x7C up to 0x7F but may be changed by the user when additional hardware requires this address. More information on usage of i/o addresses can be found in the appendix of this manual.

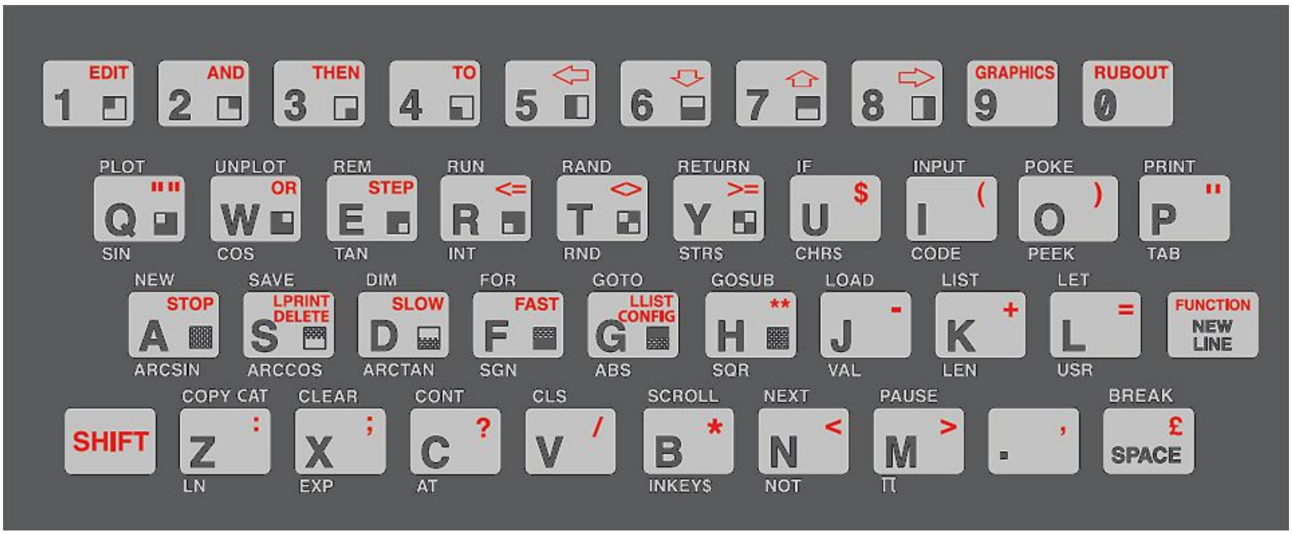
Keyboard:

The ZXmore is equipped with a keyboard with 40 keys in a matrix of 4 x 10 keys.
The allocation with keywords and functions differs from the chosen ROM in the instance (for example ZX80 or ZX81) und is only for the letters, numbers and some special chars identical.
During construction of ZXmore you have to decide whether to use the ZX80 or ZX81 keyboard layout.

The firmware ZXmaster allows translation of one keyboard layout into another to use the same layout for different ROMs. Loading and starting programs do not require special keywords. LOAD and RUN are identical for both layouts. Different layout is a matter only during active programming.

Layout ZX80:

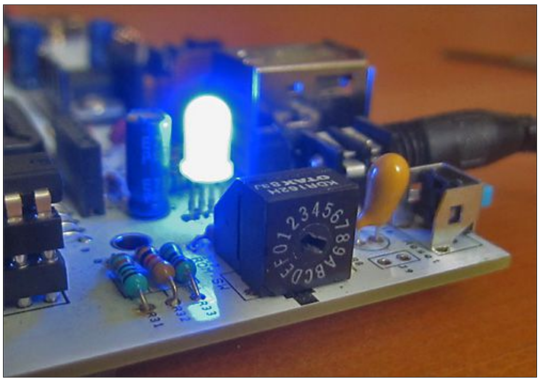
Layout ZX81:



First steps with ZXmore:

The ZXmore can be operated as desired with manual switching of the ROMs or alternatively with soft switching via the keyboard with ZXmaster.

ROM Switch



Position 1-7 choose a specific ROM firmware which must have been programmed before in the flash ROM. Only 8 positions are used from that switch, 8-F have the same function as 0-7. Position 0 selects the ZXmaster control software which allows soft switching and even multitasking on request.

Position 1-7 are useful only for ZX80 and ZX81 compatible ROMs because of a default setup after switch on for RAM and ROM size and position and some other control signals. Manual programming of the internal registers is described in the appendix.

A colored LED (rgb) shows the active instance with a different color regardless if the instance is running from the manual switch or from ZXmaster control software:

0 = dark/off
1 = blue
2 = green
3 = cyan
4 = red
5 = violet
6 = yellow
7 = white

Manual switching has the highest compatibility because the ZXmaster control routines are not used and can not disturb any unknown program with some maybe hidden feature / hardware control. But some functions are not available in manual operation liek change of keyboard, switching of instances, USB drivers for loading/saving data and some special display modes are maybe not available.

Additionally the parallel operation of several instances is not available and after every switch the system is reset and does a new power-up (clear of RAM, etc.).

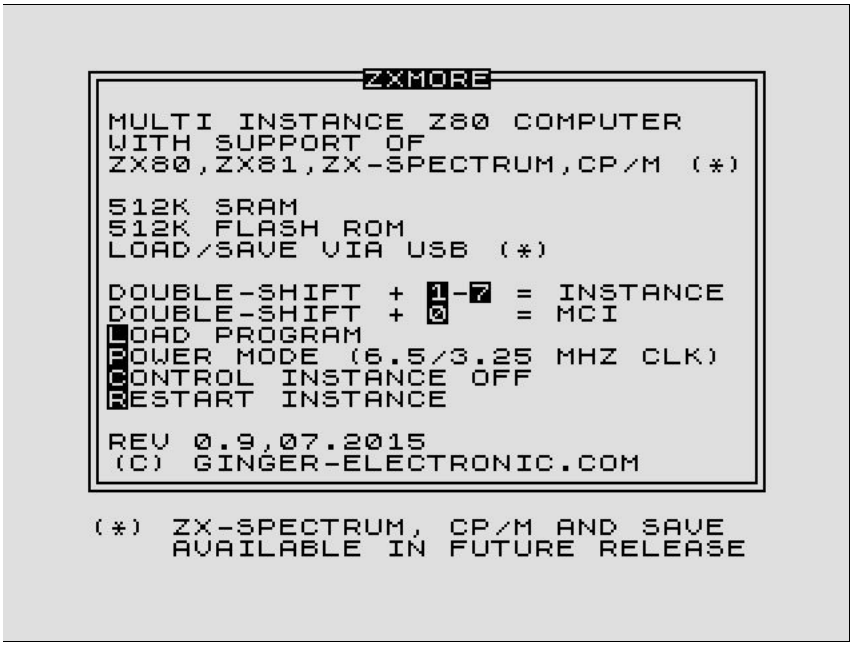
Reset switch

The reset switch functions different depending on the context used. In manual operation (position 1-7) the switch does a cold start of the running instance. When pressing in position 0 the ZXmaster switches back to instance 0 and does a warm start only. In this context the instance information, status, RAM contents, etc. are not destroyed and can be continued when switching back to these instances later. The last running program will be continued.

It is possible and likely that the last running instance where reset was pressed is lost or crashed as the last status could not be catched by the ZXmaster firmware and a restart may fail. But all other instances are „frozen“ and can be continued after switch back safely. Due to the concept with own and dedicated RAM areas the other instances can not be disturbed under normal circumstances.

Handling with double-shift (DS)

After switching on you should be welcomed with the start screen:



All control commands of ZXmaster function with a double-shift key (DS shortened in the manual) by shortly pressing the shift key twice consecutively and a second key after (showed invers in the start screen.

The key have to be pressed short and fast – the maximum timeout between two keys may not exceed 1 second for save detection. The reason for this control keys is that the normal ROMs like ZX80 or ZX81 wouldn't influenced with this key combination (double-shift has no special meaning).

All double-shift keys are processed from the ZXmaster only and not detected by the running instance.

Switch to instance (DS-0 to DS-7)

With DS and 0 to 7 you can switch to the desired instance. If the instance is first used it will be automatically initialized and a second call will just continue the program in the instance. The firmware ZXmaster saves the information for all instances and stores all registers in memory and the instruction pointer.

Instances can be interrupted any time by switching to another instance and are continued after switching back. Technical the switching is based on NMI which are catched automatically from instance 0 which handels all necessary action. Even the keyboard processing is done by instance 0 only and will be injected to the current instance.

Reset, warm boot, cold boot (DS-R)

Additionally to the reset switch it is possible to perform a reset by software through pressing DS-R key combination. Used in instance 1-7 the current instance will be restarted. Pressing the reset switch will do a change to instance 0 but only with a warm boot. There are circumstances where a cold start is required which can be forced by pressing DS-R in instance 0 which will reset all other instances as well.

A short interrupt of the power supply results in a warm start only. There maybe situations where an instance may hang or crashed and pressing DS-R for a single instance won't work because the display file is corrupt for example. In this case the instance 0 doesn't get control back and all double-shift commands won't work anymore. It is recommended to press the reset key followed by a DS-R combination in instance 0 to solve this.

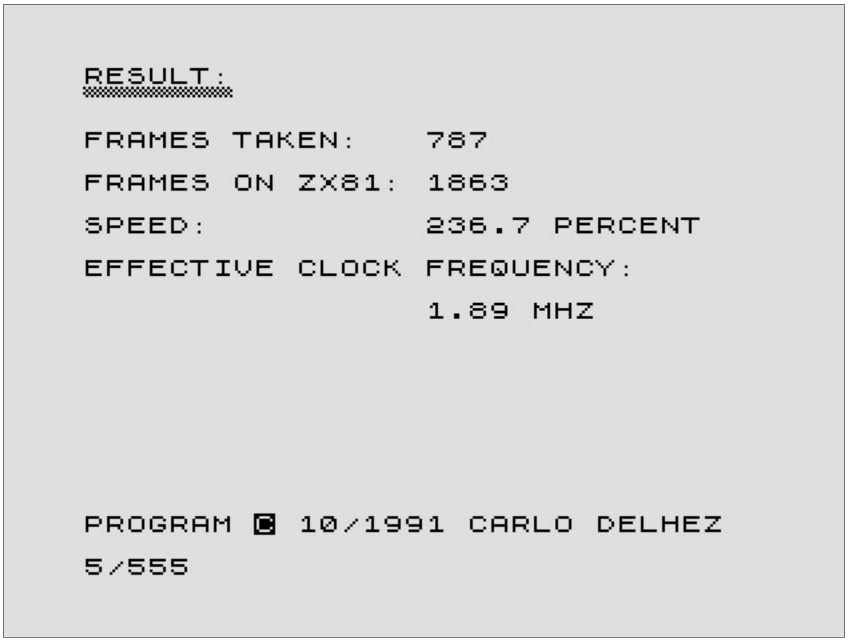
Power Mode (DS-P)

The default clock speed is 3.25 MHz for ZX80 or ZX81 hardware.

The effective clock speed is much lower with approx. 0.81 MHz because the display routines require much time of the cpu and the user program is interrupted many times.

The ZXmore can compensate this disadvantage by doubling the clock speed outside the video routines to 6.5 MHz. The resulting speed increases a bit more than double due to some more hardware optimizations.

The imaged screenshot shows the unofficial benchmark for ZX81 systems and the speed increase with ZXmaster in version 0.9



Sometimes a program may not work correctly or maybe too fast. If this annoying the speed can be changed to 3.25 MHz only which is handled per instance. It is possible to active a fast and a slow instance for example. Every DS-P key combination changes speed in current instance from low to fast or fast to low consecutively.

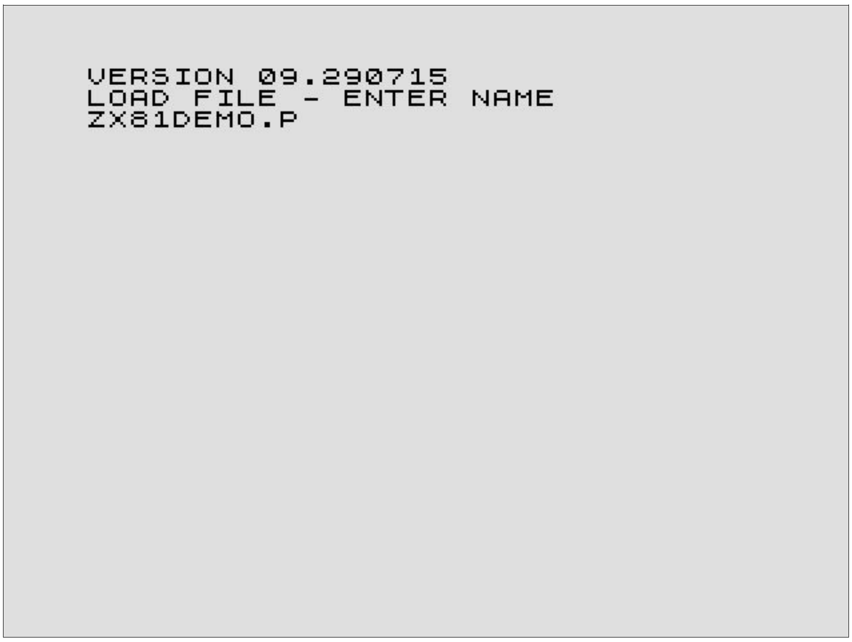
Compatibility mode (DS-C)

The firmware ZXmaster supports the smooth operation of ZX81 instances. There may be situations where a program used with ZX81 does not behave in the same way as on real hardware. The compatibility mode may be chosen by the user in these cases to get a most compatible environment by the price of losing control through ZXmaster which can be reactivated by pressing the reset switch when needed. Especially HRG programs executed in version 0.9 may not display an image as the video handling is different.

Whenever programs do crash or not behave in the same way as expected there may be a good choice for the compatibility mode for these programs to run better. If the speed should be decreased or a program loaded via USB flash disk this can be done first and then pressing DS-C after.

LOAD via USB (DS-L)

The USB loader can be started with DS-L from any instance (in version 0.9 only for ZX81).



The loader message appears with its version and the name of the desired program can be typed in. In the current release the loader works only for files in the root system and 8.3 format file names (no LFN support in this release).

The loading speed is about 150kByte / second and due to technical reasons in FAST mode only. So the screen may flicker shortly when loading data or programs. The USB flash medium should be formatted with FAT32 and file names my entered with chars, digits and the point separator for file names only.

Even long programs should be loaded instantly or at least with a delay of a few hundred milliseconds only because programs normally should be smaller than 16kByte. So loading will be finished in pretty less than a second. The very first acces on a new inserted USB flash medium/drive may have a bit more latency.

If the requested file could not be found the error code 65535 will appear (maybe just a typo). If the USB medium/drive is not inserted or maybe wrong formatted, the error code 65534 is displayed. The loading can be aborted by pressing DS-L again in the enter mode. The displayed error has to be confirmed by pressing any key.

If the instance 0 is called after loading data in another instance there may be displayed some status information like the loaded filesize.

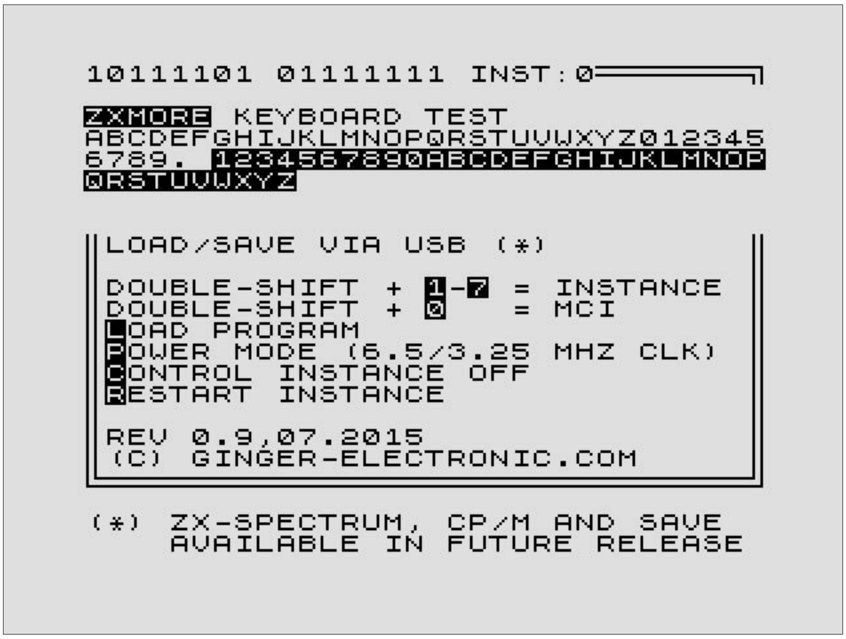
Debug functions:

Test mode (DS-T)

ZXmaster does not respong to normal key presses but to double-shift combinations. With DS-T a test mode can be activated for keyboard testing.

The proper function of any key is possible in this mode which will just overwrite the screen with the pressed character. Only the base characters are printed like 0-9, A-Z, dot and space.

A press on NEWLINE will put a dash on the screen (-) and SHIFT will print the inverted char. Additionally the scan code of the key is displayed in the first screen line plus some status information.



USB load test (DS-L)

In instance 0 the load command does not load a file into the instance memory as program but shows the information found while displaying the first and the last 128 chars as hex values on screen.

First the status code is displayed (FF00 for file found on disk) followed by the size in bytes and followed by the first and last datablock.

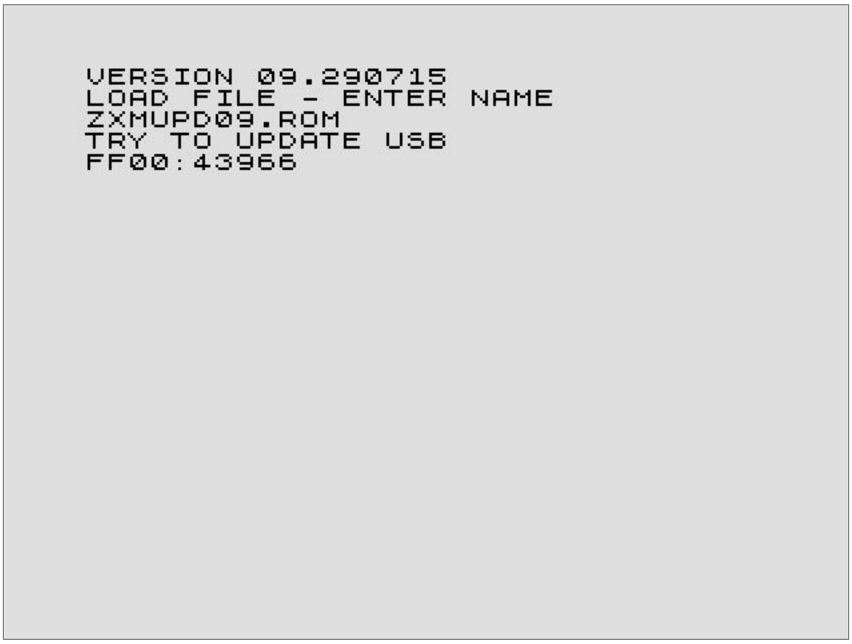
Possible timing or loading problems could be investigated with this feature. Normally a file will end with a $80 char as mark of End of VARS section (E_LINE).

Update function:

As soon as an update for ZXmaster is available or a new USB firmware to be loaded onto the FTDI chip it should be installed for further use.

Update USB firmware

The USB chip can be updated or maybe downgraded with a special file on the USB flash drive.



These update files can be stored permanently on the stick and are installed only manually on request by activation the loader with DS-L and entering the corresponding ROM file name. This works only in instance 0.

Update files are named with ZXMUSB plus a version number and file extension „.ROM".
Wrong rom files will be ignored from the USB chip and not loaded.

After starting the update procedure you should wait for at least 10 seconds and do nothing further – the USB chip will be flashed/reprogrammed in this time period. After 10 seconds the ZXmore should be switched off for at least additional 10 seconds (by switching off power – not just pressing reset) and the system will be ready to use after startup.

If the update succeeded a new version message will be displayed when starting the loader again with DS-L. The first two digits show the version number, the following 6 digits show the version date.

Attention:
It is recommended to use a dedicated USB flash medium only with the ZXmore or ZXmaster which does not contain important or valuable data which is not backed up first on a safe place. You may find additional information in the appendix with legal information and assumed liability.

Appendix

Technical data ZXmore (hardware)

- CPU Zilog Z80 (8 bit)
- flash ROM 512 kByte, organized as 8x 64kByte
- SRAM 512 kByte, organized as 8x 64kByte
- variable memory layout adjustable in 4kByte steps
- ROM/RAM swap mode (for CP/M operation e.g.)
- optional write protect for ROM
- RAM mirror defeatable (A15) with 56 kByte RAM in ZX81 mode
- extended M1NOT mode with assembly programs in entire 64k memory
- HRG in entire memory (RAM and ROM)
- UDG in entire memory, 64 and 128 char set mode
- video char mode with 24 x 32 chars
- video HRG mode with 256 x 192 pixels
- 6.5 / 3.25 MHz clock frequency, individual preset per instance
- 2x USB host port for flash drives and peripherals
- real time clock (RTC)
- 2 x Audio 3.5mm audio jack (EAR/MIC)
- expansion connector with 2x 22 signals for ZX80 or ZX81 hardware extensions
- power supply 9V, ca. 100mA current consumption (without USB devices connected)

Features ZXmaster firmware

- multi instance system, up to 7 different operating systems concurrently
- multitasking mode with as many instances as desired
- ZX80 and ZX81 compatibility
- prepared for ZX Spectrum (monochrome) and CP/M
- updateable USB drivers
- Z80 boot mode with ZXmaster firmware installation/update
- flash programming for individual and permanent configuration of instances

Hardware concept of ZXmore

The following chapters explain the architecture of ZXmore seen for different hardware aspects. The hardware is compatible to ZX80 and ZX81 but not identical. The following paragraphs show the differences in concept.

Memory concept:

The ZX81 and ZX80 are based on mirrored RAM and ROM.
The typical segmentation of a system with 16kB RAM:

```
$0000-$1FFF ( 8kB ROM)
$2000-$3FFF ( 8kB ROM mirrored from $0000-$1FFF)
$4000-$7FFF (16kB RAM)
$8000-$BFFF (16kB mirrored ROM)
$C000-$FFFF (16kB mirrored RAM)
```

It is possible to switch off this mirrored sections but the mirror at memory address $C000-$FFFF is required for the video display system.

The ZXmore in comparison can do the video display without mirrored memory areas. You will find more information in the section video display system. ZXmore has 64kB ROM and 64kB RAM which addresses overlapped partly. There is a border between ROM and RAM which can be moved in steps of 4kB in any direction.

The memory layout can be defined different for any of 7 instances. It is possible to move the border on request to access some hidden memory areas to place additional drivers in ROM area which use only sparse RAM and can release RAM areas after use. On the other hand it is possible to use additional RAM for storing temporary data when hide ROM areas.

A typical memory layout could be:

```
$0000-$1FFF =  8kB ROM
$2000-$3FFF =  8kB additional ROM/drivers
$4000-$FFFF = 48kB RAM
```

or

```
$0000-$1FFF =  8kB ROM
$2000-$FFFF = 56kB RAM
```

or in A15 compatibility mode (see video display system)

```
$0000-$1FFF =  8kB ROM
$2000-$3FFF =  8kB ROM or RAM on request
$4000-$BFFF = 32kB RAM
$C000-$FFFF = 16kB mirrored RAM (from $4000-$7FFF)
```

More details can be found in section „Hints for programming the latches".

Video display system of ZX80/ZX81:
Video display is based on techniques of ZX81 with using the Z80 processor in conjunction with a shift register for printing pixels on screen without additional video processor hardware. This saves monetary cost but the price will be paid by a slowed down system with approx. 25% cpu time for the application. The resulting clock frequency will be only 0.81 MHz from real clock frequency of 3.25 MHz.

During video display the so called display file will be executed as code but only instructions with bit 6 set (like HALT) – all others are used for addressing the char generator. To distinct video code execution from normal code execution the address line A15 has been used in conjunction with M1 for activating video processing hardware in the original ZX81. Consequently programs with machine code / assembly can not be executed in memory areas above address $8000.

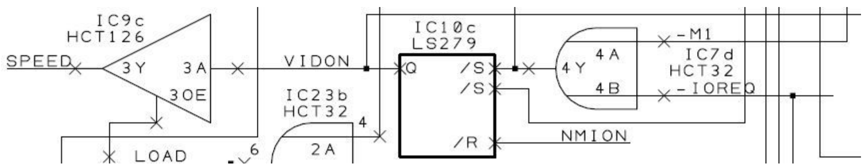
If a 32k RAM extension is used with a ZX81 only the first 16k may be used for assembly code and the second 16k above this address only for storing data. A plain BASIC program with 32k is possible as it consists normally only of data structures to be interpreted from ROM routines. One more important thing is, that the display file may not cross the $8000 border when using 32k programs.

Widely used is the M1NOT modification which restricts the video code area to $C000-$FFFF together with A14. Disadvantage is that the display file can not be in area $8000-$BFFF when using 32k BASIC programs. The second 16k are used mostly for assembly code / drivers only.

There are a few solutions with up to 56k RAM which activate memory mirroring during execution of the display file only and switch off mirroring for data usage (read/write no execute). This couldn't be used for code either but for high resolution graphics (HRG) due to its big size of 6kB for 256x192 pixels.

Video display system of ZXmore:
The video code execution detection of ZXmore works different and is not bound to a specific memory area. It is an extended M1NOT mode which allows execution of assembly code in the whole memory ($0000-$FFFF) regardless if code is in ROM or RAM.

The ZXmore uses the HALT synchronisation mechanism which is used together with the (last) NMI in the margin area. This synchronisation is used from nearly any program even with own video routines or high resolution graphics. Additionally video mode is detected when interrupts occur (interrupt acknowledge cycle) which are useful for video routines only due to A6 bound.

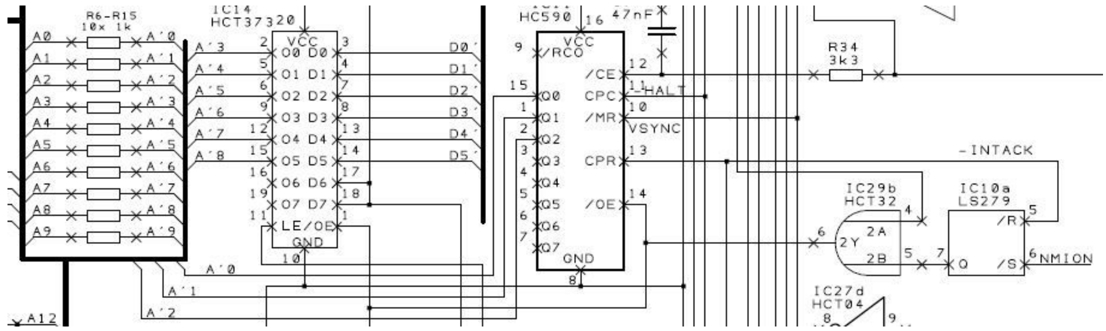


As this detection is unique there is no special use of A15 or any other address needed. Outside of video display machine code can be executed on any runtime address. The clock frequency is reduced automatically to 3.25 MHz during video mode to assure the correct display timing. The end of video display is detected by activation of NMI again.

HRG and UDG:

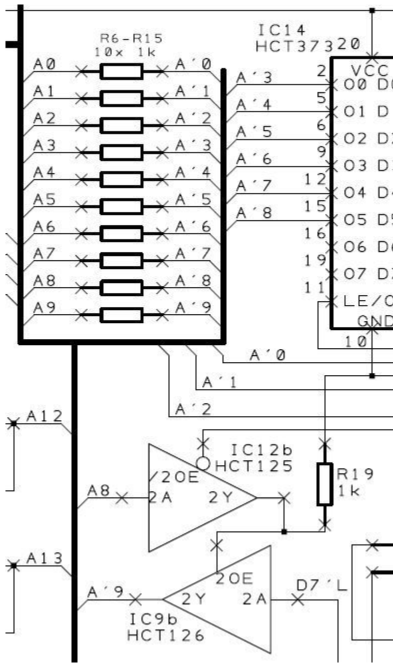
The ZX81 uses a character generator which automatically overdrives the address bus for feeding the shift register with data from the char map inside the ROM in conjunction with a scanline counter (0-7). The address is derived from the index register, the char read in the display file and the scanline counter.

This overdrive is used for the ROM addressing only (through the ULA). Consequently char maps can be place in ROM only and are not usable from RAM for user defined graphics (UDG). On the other hand high resolution graphics (HRG) can be used with RAM only to read the pixel file (256x192 pixels in 6kB area). So HRG works only from RAM and UDG only from ROM on a standard ZX81 hardware.



The ZXmore has no direct assignment of memory areas for HRG and UDG. The address override is activated with interrupt acknowledge cycles only which are used with UDG only but not with HRG. Plain software HRG which address the content for the video shift register with index register and refresh register do not use any interrupts.

This circumstance with no interrupts when using HRG is used to control the address override mode. Any INT ACK sets a flipflop which activates the char map logic while activating NMI resets this flipflop for index/refresh register addressing.



So user defined graphics may be stored in ROM or RAM and are addressed with index/refresh register. It is possible to use 64 char sets (with 64 inverted chars) or use of 128 chars while the second 64 chars are selected with D7 which is normally used for just inverting the chars.

Bit 7 automatically overrides A9 during char map mode as soon as bit 0 of index register (A8) is set during the refresh.

A normal char set consists of 64*8 bytes (=512 bytes) and addressing is used at even boundary only – so bit 0 of index register is used to control the usage of 64 or 128 char sets:
0 => 64 chars (512 bytes)
1 => 128 chars (1024 bytes)

Keyboard:
ZXmaster does not us the keyboard driver of ZX80 or ZX81 but an own driver with a faster debounce mechanism and some more features like key repeat. This is useful when a long input line should be aborted while deleting the chars faster than usual. When pressing any key down longer than 1 second the key will be repeated processed for the duration of the keypress with approx. 25 strokes per second as long as the ROM is able to handle them fast enough due to missing keyboard buffer.

The keyboard driver additionally handles the double-shift commands and catches them in a hidden way to complete the desired action in the background.

There are more keyboard features planned with later versions of ZXmore like key rollover for faster key processing (especially beneficial for CP/M), a repeat of the last entered input lines and definition of keyboard macros.

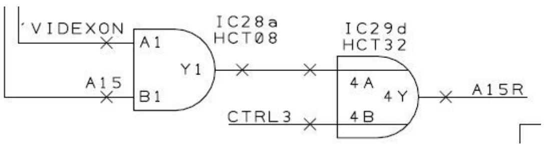
NMI control and WAIT:
The ZX81 requires regularly NMI pulses which decrease a software counter (register AF') and activate the video display or keyboard processing with vertical sync when NMI period expired. NMI's are executed every 64us when horizontal sync occurs.

Depending on the video mode the application program is executed 2x 55 lines (PAL) or 2x 31 lines (NTSC) per frame (50 times per second). A lot of time is waisted for execution of NMI's synchronized with a WAIT period to perform synchronisation. This synchronisation would be necessary in the last NMI line only but has been implemented on every NMI due to technical simplification.

To enable the control of other instances and inbuilt functions the ZXmore catches those NMI's from other instances. The WAIT cycles are executed only in the last NMI line which gives a speed increase of about 10% even at low clock frequency of 3.25 MHz.

The interception allows control of ZX80 or other system ROMs which don't use NMI handling at all – for example ZX Spectrum or CP/M as well. Switching a ZX81 instance deactivates the control functions of ZXmaster temporarily but will be activated again when NMI switched on again (for example with SLOW statement). This way it is possible to use the standard LOAD routines using EAR plug for loading programs from audio cassettes.

A15 mode:
ZXmore does no mirror of RAM at default. Access to the display file with A15 set is redirected to address $4000-$7FFF while resetting A15. This does not work with 32k BASIC programs with display file above $8000.



In this case the A15 mode can be activated which disables mirroring of RAM and limits the available total RAM to 32k only. CTRL3 signal fixes A15 to high which provides 32k RAM from $4000-BFFF while mirroring $4000-$7FFF to $C000-FFFF. The display file can stay in area $4000-$7FFF or $8000-$BFFF in this mode.

Hints for programming the latches

I/O addresses:

I/O addresses are incompletely decoded due to compatibility to ZX81. I/O addresses use exclusively one from 8 bits (low active) which reduces the capability of the system to handle 8 devices only.

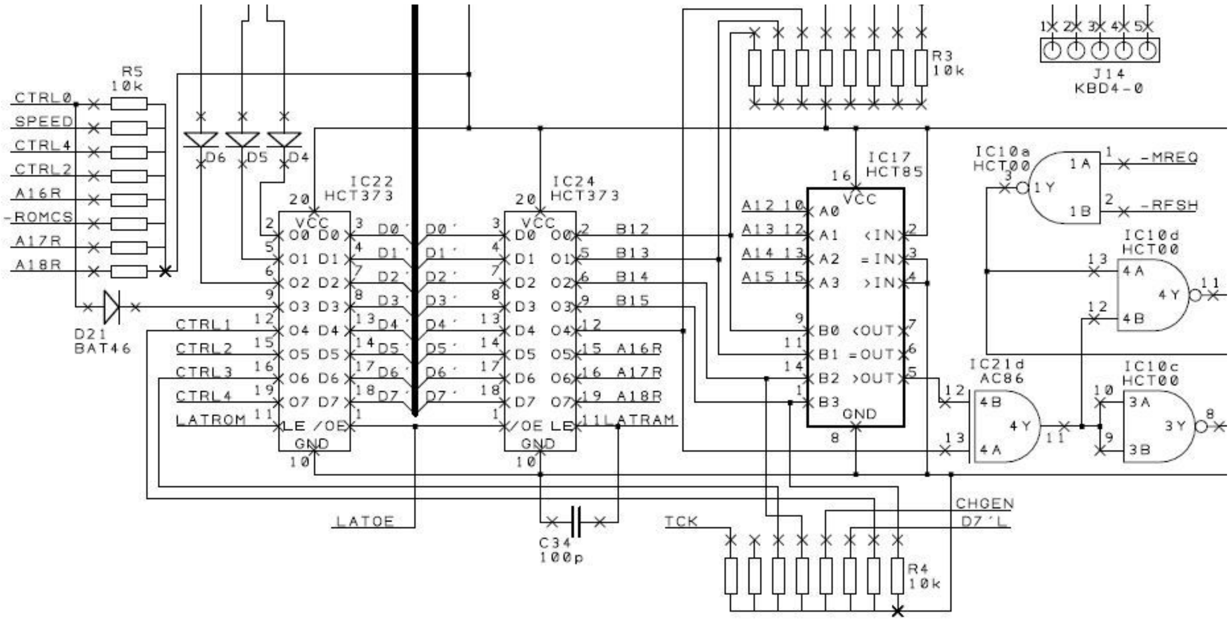
Following addresses are used from the ZX81 hardware:

A0 address $FE => switches on NMI circuit and keyboard handling
A1 address $FD => switches off NMI circuit
A2 address $FB => control of printer

For the additional features of the ZXmore for address the latches and USB communication a further address bit is used which may be choosed from A3/A4/A5/A6/A7 for avoiding address overlap when additional hardware modules are used. Standard address together with A0 and A1 is:

address $7F USB communication
address $7E RAM latch
address $7D ROM latch
address $7C internal usage

Interesting for programmers may be the RAM and ROM latches to control the memory management. This is recommended for experts only as this may influence the stability of the ZXmore.



The schematic shows the latch control. The RAM latch controls an address comparator which compares the programmed address with the current address and activate RAMCS when the current address is greater than the programmed and otherwise activates ROMCS. Bit 4 allows to swap ROM and RAM.

Bit 5-7 selects the memory page (64k) used for the active instance while these bits are inverted. Instance 0 has all 3 bits set while instance 7 has all bits reset.

Following example:

Instance 4 should be started with 16k ROM and 48k RAM.

The RAM latch needs to be programmed with $73:

instance 4= $60

first ROM, then RAM =$10

ROM > 12k (=16k) = $03

Extract of ZXmaster firmware as example:

```
define  LATRAMSTARTUP   $F1     // default value RAM latch ZXM (16k ROM, 48k RAM)
define  LATRAMZX81      $F1     // default value RAM latch ZX81 (16k ROM, 48k RAM)
                                // bit 0-3 ROM size in 4k steps, 4k-60k ROM
                                // bit 4   0=RAM low/ROM high, 1=ROM low/RAM high (swap RAM/ROM)
                                // bit 5-7 RAM page inverted, (0=7 and 7=0)
```

ROM latch control:

```
define  LATROMSTARTUP   $3F     // default value ROM latch (speed, multitasking, 48k)
define  LATROMZX81      $3F     // default value ROM latch (speed, multitasking, 48k)
                                // bit 0-2 ROM page inverted (0=7 and 7=0)
define  BIT_WRPROTECT   $08     // bit 3   CTRL0, 0=no write protect, 1=write protect flash ROM
define  BIT_SPEED       $10     // bit 4   CTRL1, 0=normal speed, 1=double speed
define  BIT_EXTCLOCK    $20     // bit 5   CTRL2, 0=external clock, 1=internal clock
define  BIT_A15MIRROR   $40     // bit 6   CTRL3, 0=no RAM mirror (48k), 1=RAM mirror (32k)
define  BIT_MULTITASK   $80     // bit 7   CTRL4, 0=control image active, 1=control image off
```

An example to use:

In adress area $8000-$FFFF additional ROM code is placed which should be executed. The standard RAM should be available for use. This change is possible from RAM only (driver code).

First the control instance should be switched off with setting bit 4 in ROM latch. This is important as the next NMI would change to the default latch configuration which would result in a crashed instance. For this example we are using instance 1 and have to program $E7 in RAM latch. This give ROM=$0000-$7FFF and RAM $8000-$FFFF but due to bit 4 reset RAM and ROM are swapped, giving RAM from $0000-$7FFF and ROM $8000-$FFFF.

When finished the default registers should be restored and the control of ZXmaster should be activated again through setting bit 7 in ROM latch (CTRL4). For these adhoc access the program has to know in which instance it is running and with which options it was started. Even video display would still work in this example.

These features are for advanced assembly programmers only with reliable knowledge of the hardware. This way it would be possible to use ROM banking and use of ROM code from other instances and direct access to several 100 kB of ROM code. RAM banking is much more complicated because 2 ram instances can not be used simultaneously and copying data from one instance into another could be extremely tedious.

More promising would be to write the data on a temporary file on the USB flash drive and read it back from another instance.

Legal notices

ZXmaster is intellectual property of ginger-electronic.com and may be used for non commercial purposes on the ZXmore hardware.
More information under http://www.ginger-electronic.com

open80 is a free software for the computer ZX80 published under GPL v2 licence.
More information found at OpenSE BASIC at sourceforge.net or directly under
http://sourceforge.net/projects/sebasic/files/open80/

open81 is a free software for the computer ZX81 published under GPL v2 licence.
More information found at OpenSE BASIC at sourceforge.net or directly under
http://sourceforge.net/projects/sebasic/files/open81/

„Sinclair ZX Spectrum“ is a registered trademark of
Sky In-Home Service Limited, Isleworth Middlesex, GB

Disclaimer

ginger-electronic.com is not liable for damages on hard- or software or financial loss caused by using ZXmore or ZXmaster or further supplied software or programs as long as the damage or loss is not affiliated to a reckless act or intention.

Especially ginger-electronic.com is not liable for possible loss of data when using USB flash disks oder drives. The user is responsible for saving important or valuable data and is prompted to use a dedicated separate medium with the ZXmore and ZXmaster.

Munich, 14.08.2015

ginger-electronic.com

GINGER ELECTRONIC