

Z80-MBC2



USER MANUAL

Homemade 8MHz Z80 SBC, 128kB banked RAM, RTC, SD (HD emulation), Basic and Forth interpreter, CP/M 2.2 and 3, UCSD Pascal, Fuzix and more...

D081023

Document Reference: D081023
HW Reference: A040618
IOS Reference: S220718-R290823

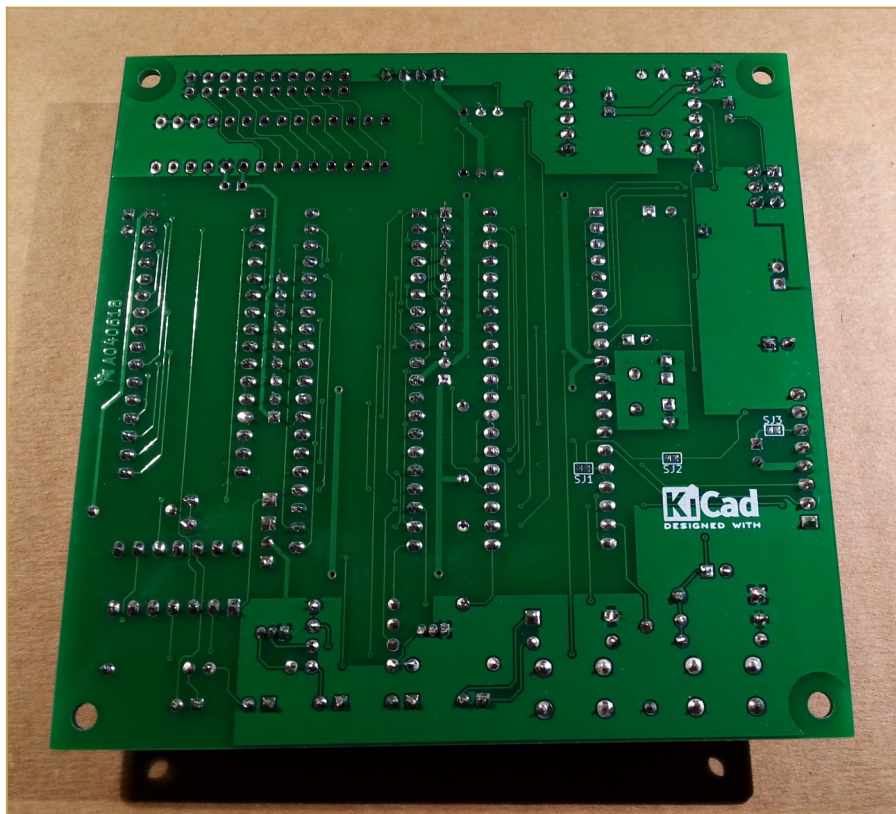
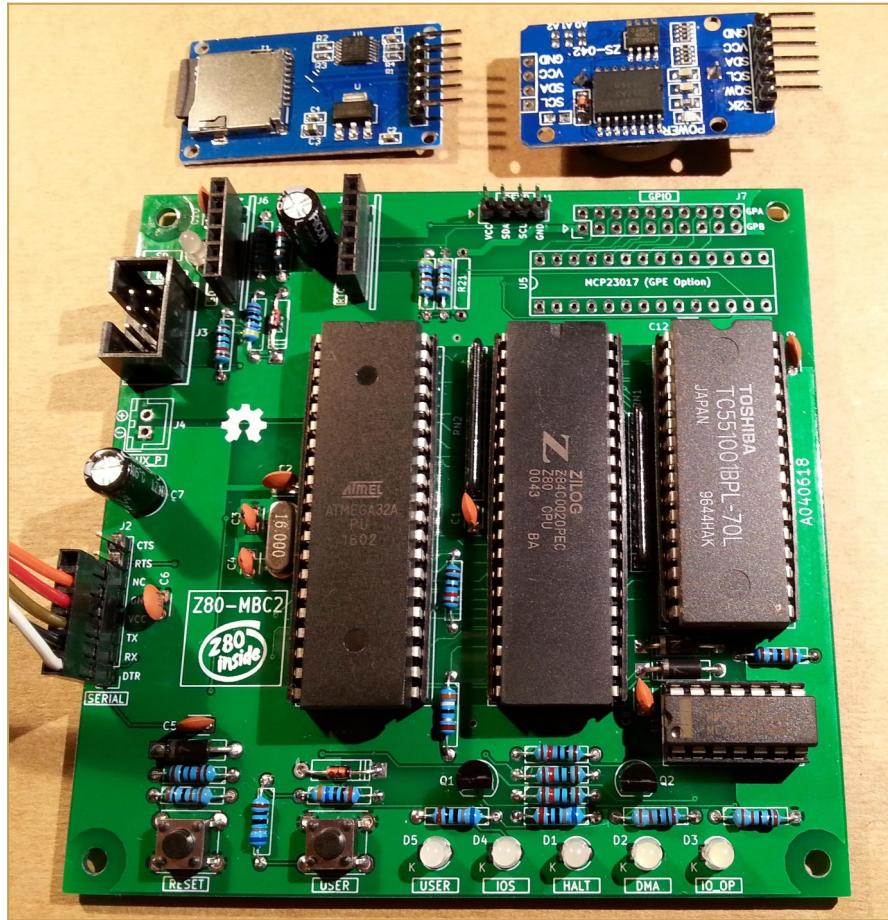
CHANGELOG:

D081023 First revision.



Table of Contents

| | |
|---|----|
| ** HARDWARE OVERVIEW ** | 6 |
| NOTES ABOUT THE COMPONENTS | 6 |
| THE GPE OPTION | 7 |
| THE SERIAL PORT | 7 |
| THE OPTIONAL RTC MODULE | 9 |
| THE OPTIONAL SD MODULE | 11 |
| ** uTERM, VT100-LIKE TERMINAL FOR THE Z80-MBC2 ** | 13 |
| ** uCOM, RS232 FOR THE Z80-MBC2 ** | 16 |
| ** SPP (STANDARD PARALLEL PORT) ADAPTER BOARD ** | 19 |
| SPP: HOW TO BUILD IT | 20 |
| SPP: HOW TO USE THE SPP CP/M UTILITY | 20 |
| SPP: WHERE TO GET A PCB | 21 |
| SPP: HOW TO GET A KIT OR AN ASSEMBLED UNIT | 21 |
| ** SOFTWARE OVERVIEW ** | 22 |
| HOW FLASH THE BOOTLOADER FROM ARDUINO IDE (LINUX) | 24 |
| HOW ENTER IN THE "SELECT BOOT MODE OR SYSTEM PARAMETERS" MENU | 28 |
| THE BAUD RECOVERY VIRTUAL BUTTON | 30 |
| THE SD IMAGE | 31 |
| HOW ADD CP/M FILES INSIDE A VIRTUAL DISK USING CPMTOOLSGUI | 31 |
| HOW TO USE ILOAD | 34 |
| HOW ENABLE THE EXTENDED RX BUFFER FOR XMODEM (CP/M) | 35 |
| CP/M 2.2 | 38 |
| CP/M 2.2 WARM BOOT MESSAGE | 38 |
| CP/M 2.2 AUTOEXEC | 39 |
| QP/M 2.71 | 39 |
| QP/M 2.71 AUTOEXEC | 40 |
| CP/M 3 | 41 |
| CP/M 3 AUTOEXEC | 42 |
| UCSD PASCAL | 42 |
| COLLAPSE OS | 47 |
| FUZIX OS | 48 |
| HOW BUILD FUZIX OS FROM SCRATCH | 51 |
| ** USING THE SDCC CROSS COMPILER ** | 53 |
| SDCC: SETTING UP AN AUTOMATED TOOLCHAIN (WINDOWS) | 54 |
| SDCC: SETTING UP AN AUTOMATED TOOLCHAIN (LINUX) | 55 |
| SDCC: USING AUTOBOOT | 55 |
| SDCC: EXAMPLES | 56 |
| ** OVERCLOCKING THE Z80-MBC2 ** | 57 |
| ** USING AN ATMEGA1284/ATMEGA1284P ** | 61 |
| ** PROJECT STATUS ** | 62 |
| ** HOW TO GET A PCB ** | 62 |
| ** HOW TO GET A KIT OR AN ASSEMBLED UNIT ** | 62 |
| ** Z80-MBC2 USER GROUP ** | 63 |
| ** LICENSING AND CREDITS ** | 63 |
| ** NOTES ON THIS MANUAL ** | 63 |



* * HARDWARE OVERVIEW * *

The needed ICs for the "base system" are:

- Z80 CPU CMOS (Z84C00) 8Mhz or greater
- Atmega32A
- TC551001-70 (128kB RAM)
- 74HC00

If you want the 16x GPIO expansion (GPE option) add a MCP23017 too.

The schematic and the BOM are attached in the Files section. The MCU Atmega32A is used as universal I/O subsystem, as Eeprom, and as reset and 4/8MHz clock generator for the Z80 CPU.

Inside the Atmega32A it is flashed an Arduino bootloader taken from [here](#), and it is possible to use the Board Manager of the Arduino IDE to "import" it.

Flash the Arduino bootloader at first (with the method you prefer), next you can upload the IOS "sketch" (the I/O Subsystem that interacts with the Z80 bus and "*virtualizes*" the EEPROM and all the peripherals seen by the Z80 CPU) using Arduino IDE.

You can use the on board **ICSP** port **J3** (also called ISP port) to write the bootloader, but remember to **disconnect any other connector** when using it. Also **both SD and RTC modules (if present) must be removed** from the board when the ICSP port is in use.

As clock source for the Z80 CPU it is used the 16MHz Atmega32A oscillator, so the "**external 16MHZ osc.**" bootloader variant must be chosen when flashing the bootloader from the Arduino IDE!.

The 74HC00 is used as RS flipflop to stop the Z80 CPU during I/O operation, giving the needed time to the Atmega32A to interact with the Z80 bus, and as part of the MMU.

Note that **only the CMOS version of the Z80 CPU can be used here**. This because only CMOS version, under given condition that are respected in this schematic, has logical levels compatibles with Atmega32A and 74HC00.

NOTES ABOUT THE COMPONENTS

You should use a Z80 CMOS speed grade of at least 8MHz for full speed, but setting the clock speed at 4MHz you can use a 4MHz Z80 CMOS version too (or you can try to overclock it at 8MHz...). The 74HC00 can be substituted with a 74HCT00 if you already have one. The RAM chip TC551001-70 can be substituted with any suitable 128kB SRAM).

Please note that the **USER led * must * be blue or white** (or pink... I've some pink leds that seems to have a Vf like blue one. May be I'll do a board with them...) just to be sure

that **V(forward)** is $\geq 2.7V$ (otherwise the USER key may not work as expected).

The **J4** connector (**AUX_P**) is not currently supported and is not populated by default.

The three solder jumpers (**SJ1-3**) on the bottom side are not currently supported and **must be left opened** (as stated in the schematic).

THE GPE OPTION

It is possible to choose to populate on the PCB a GPIO port expander (U5) to add 16 bidirectional GPIO pins. The GPE option (see the schematic) can be used with the SPP Adapter board (see the paragraph: **SPP (STANDARD PARALLEL PORT) ADAPTER BOARD**).

THE SERIAL PORT

The **SERIAL** port (**J2**, see schematic) can be connected with a TTL-RS232 adapter, or with a serial-USB adapter.

I've used a serial-USB adapter that acts also as power source for the Z80-MBC, and has the **DTR** signal for the "autoreset" driven from the Arduino IDE. For a terminal that has a serial TTL port no adapter is needed.

Of course to upload a "sketch" from Arduino IDE you need to use a serial-USB adapter connected to the SERIAL port.

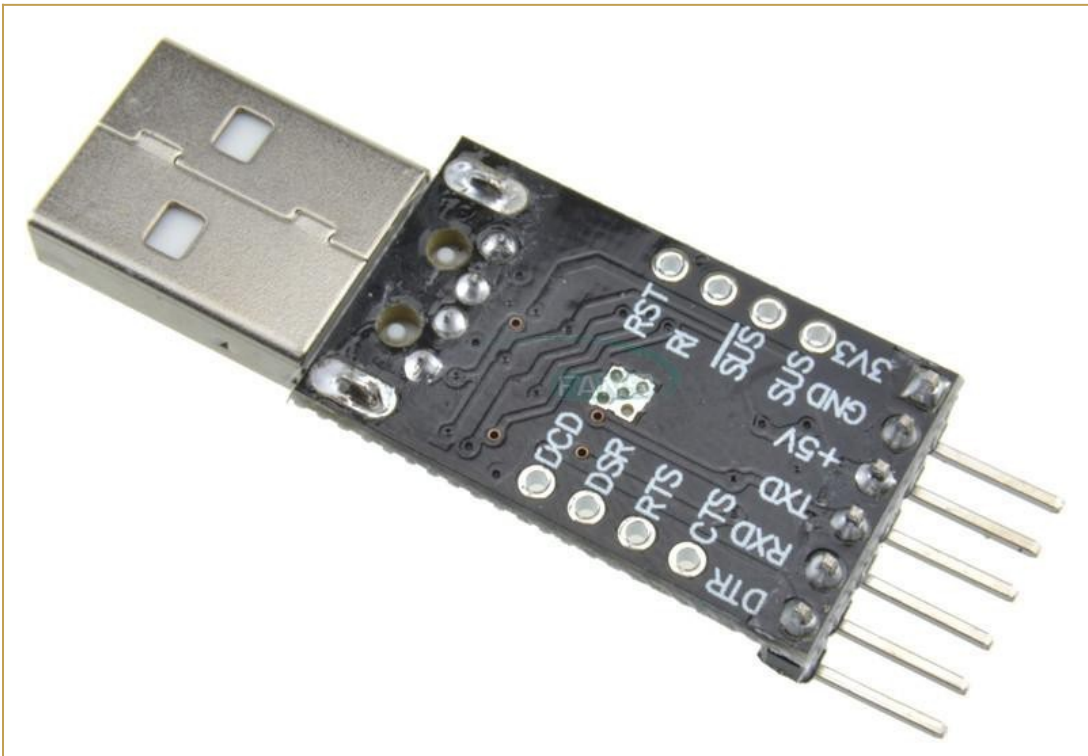
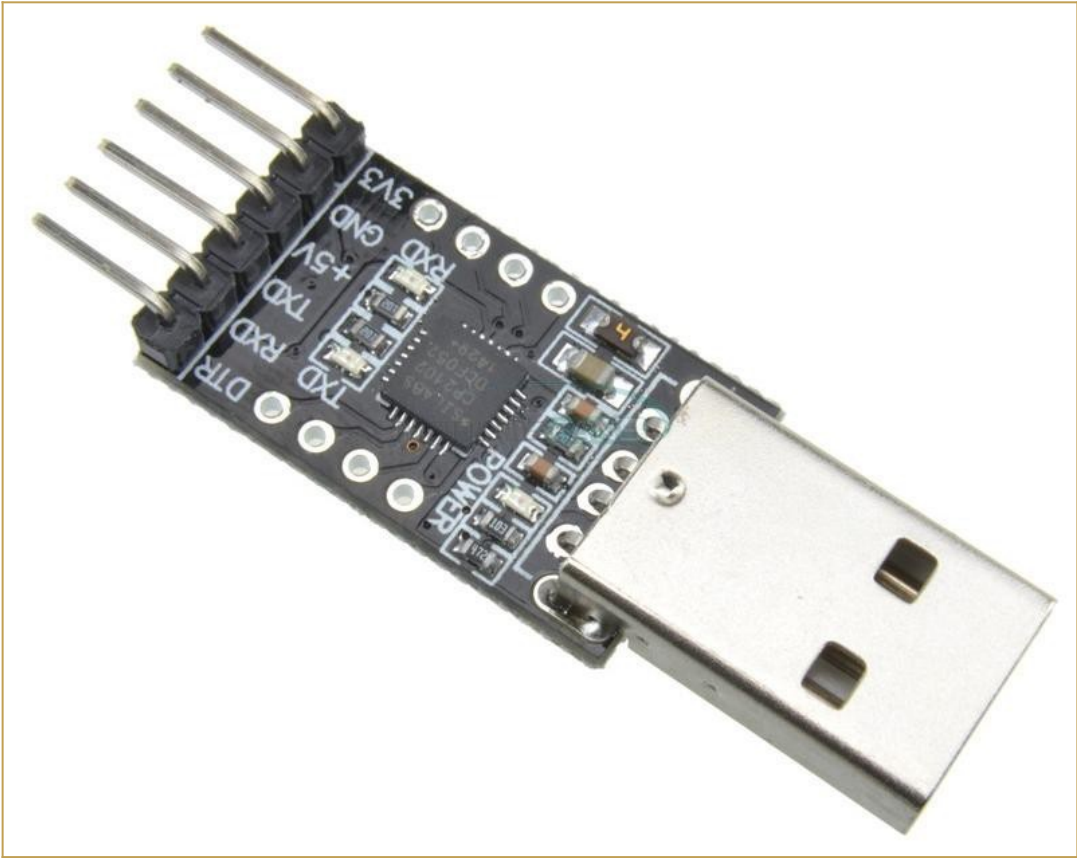
Note that the RTS and CTS pins of the SERIAL port are not currently supported and must be left not connected (as the NC pin!).

The 3V3 pin of the serial-USB adapter must be left disconnected (if present).

You should use those Serial-USB adapters that have the DTR pin on the connector. It is suggested to have also the CTS/RTS signals available for future upgrades.

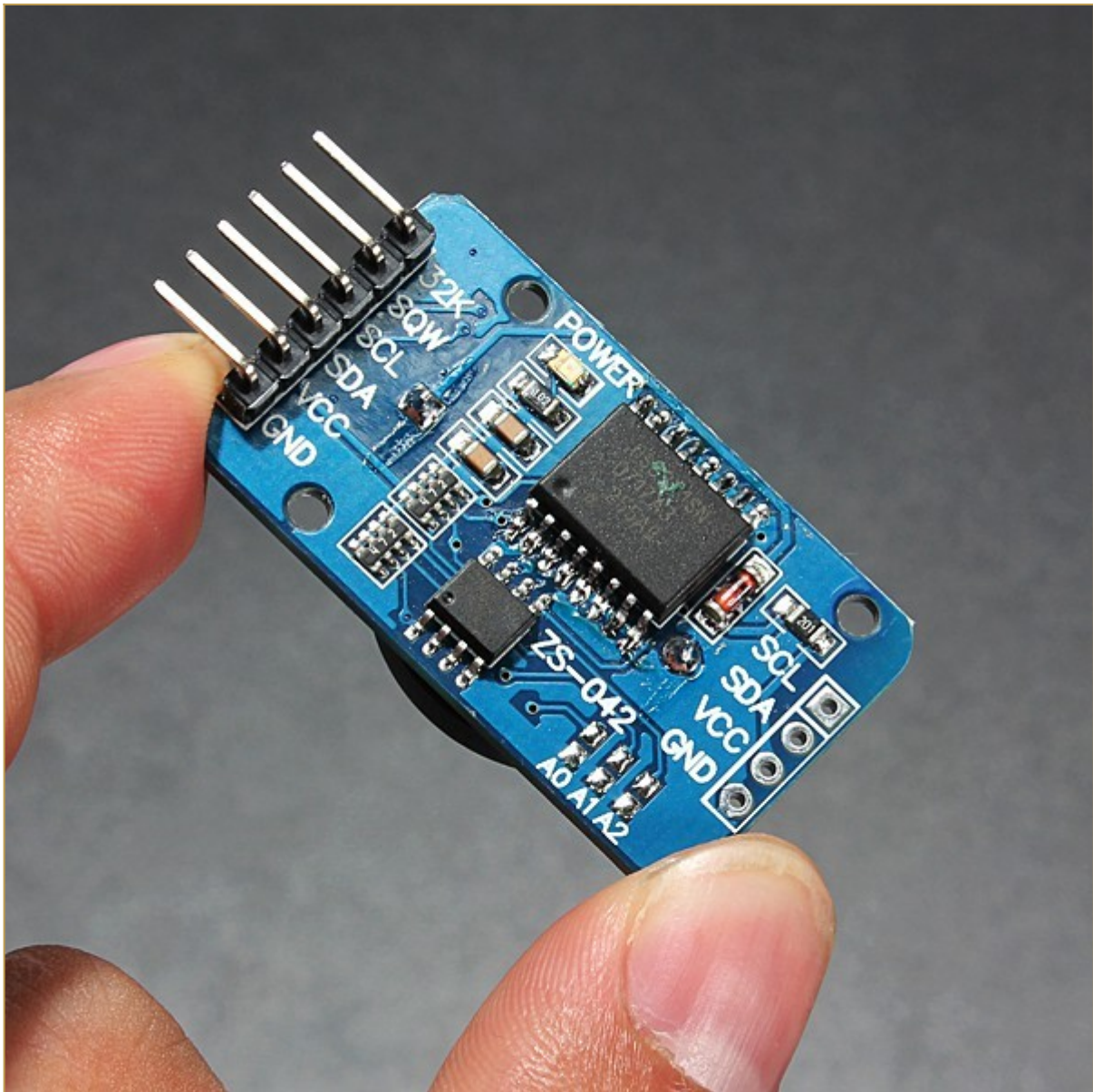
Please note that ***all the pin names of J2 on the PCB are referred to the serial-USB adapter, so all the signals as TX and RX are relative to the serial-USB adapter side (in other words TX and RX are already "inverted". See the schematic).***

Here a suggested serial-USB adapter based on a CP2102 (very common on ebay):



THE OPTIONAL RTC MODULE

The RTC is a common module based on a DS3231 RTC like this one:



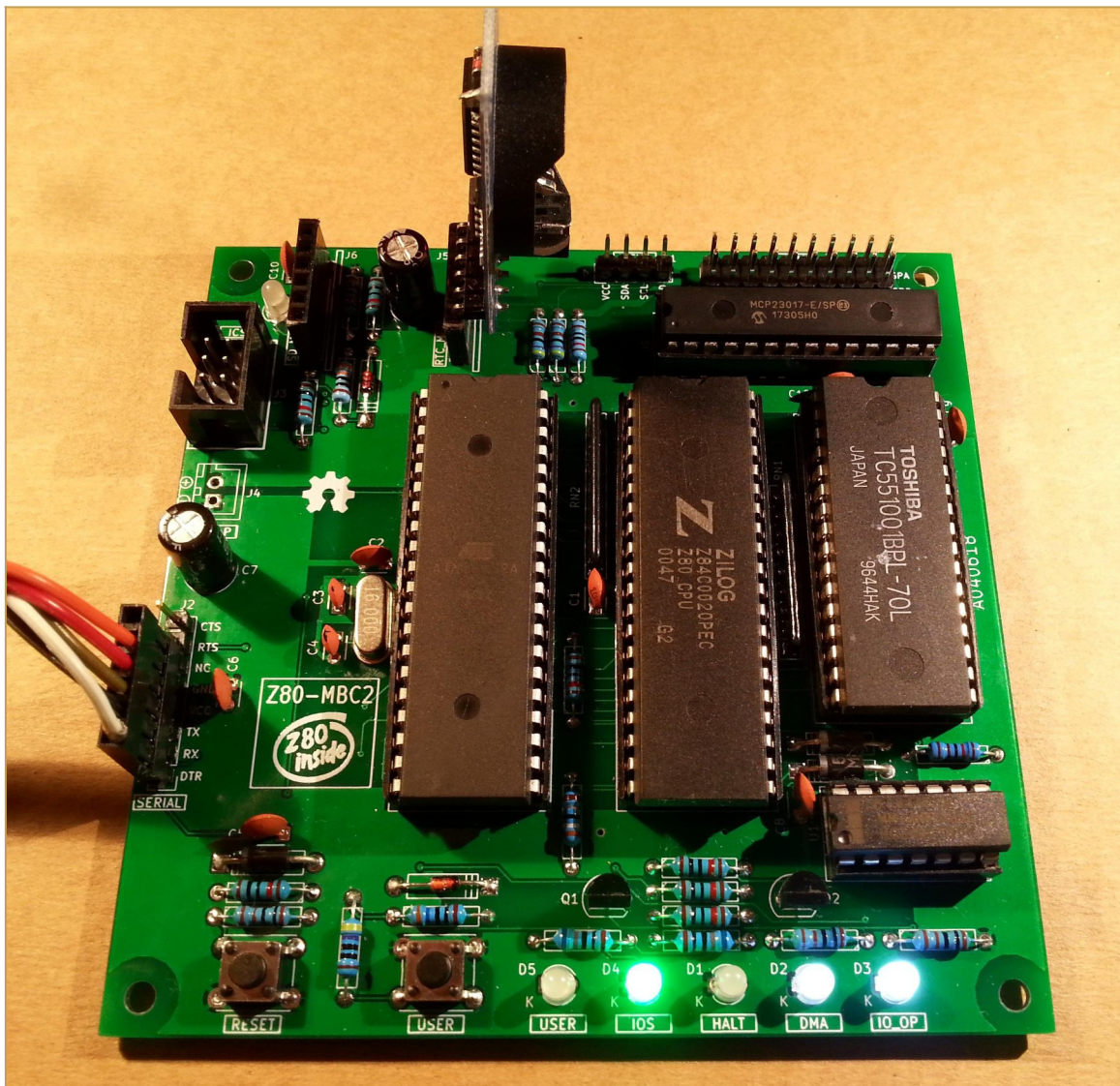
This cheap modules have a trickle charging circuit that may cause the "explosion" of the battery if you use a standard CR2032 cell. More, it can damage also a rechargeable LIR2032 cell. For more information and how to fix it see [here](#).

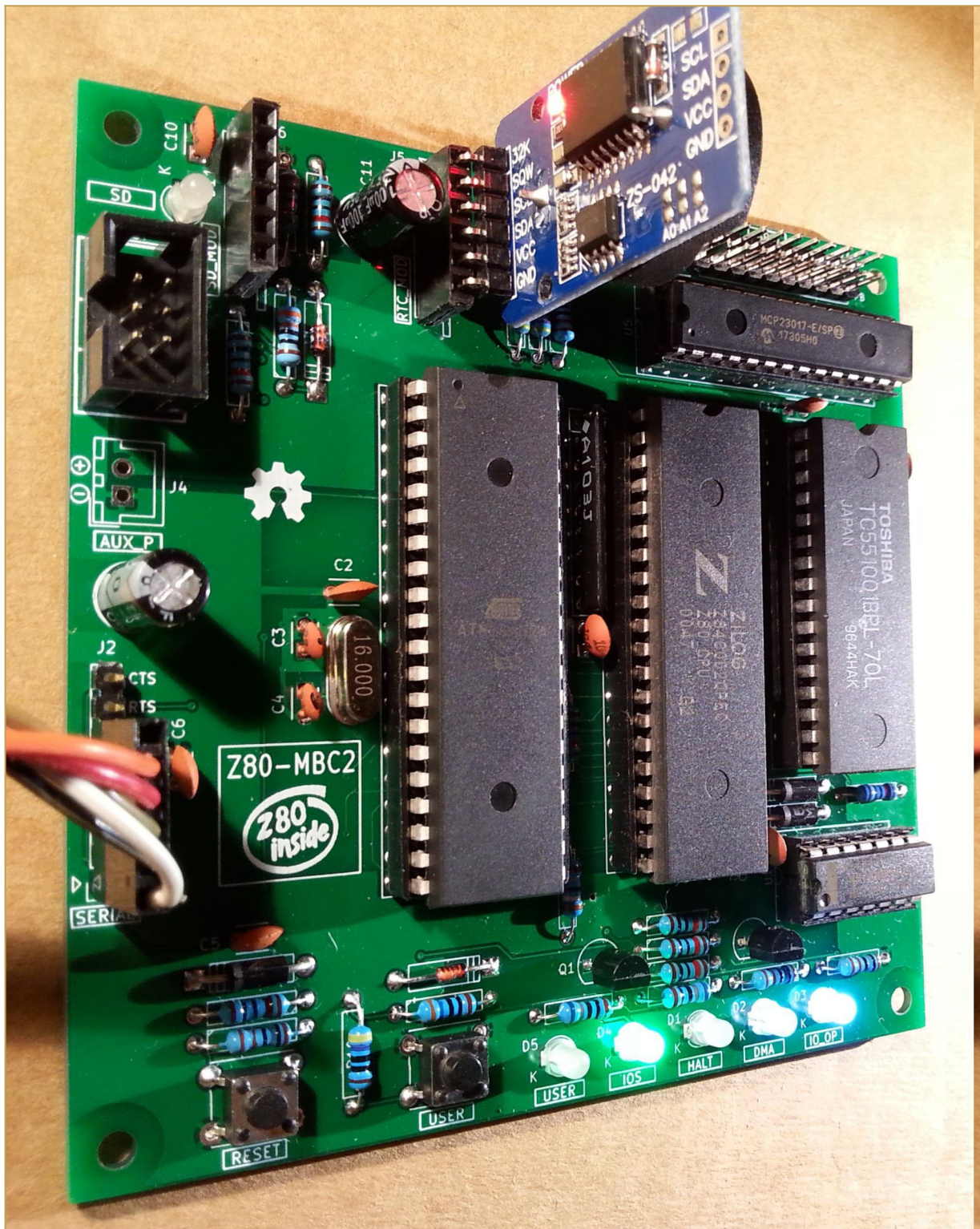
The RTC module has it's own pullup resistors on SDA and SCL. Because the value is 4k7 (the same value used inside the Z80-MBC2 board), the resulting value will be:

$$4k7 \parallel 4k7 = 2k3$$

Because this value is fine there is no need to take away the pullup on the RTC module.

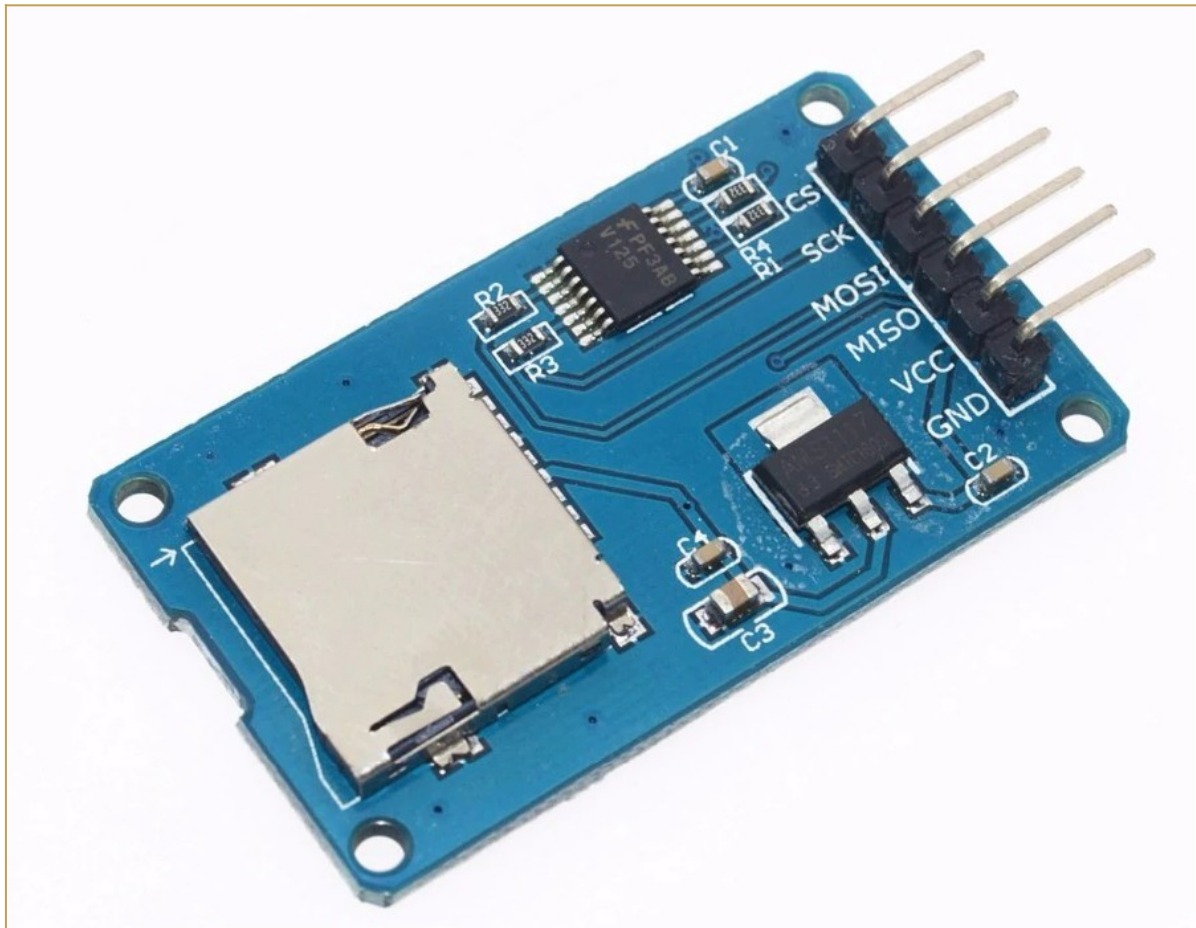
Pay attention on how and where you plug the module in (the only right connector for it is **J5** marked as **RTC_MOD**). If you plug it in the wrong connector or in the wrong way it is possible cause permanent damages to both the module and the Z80-MBC2 board! So plug it as shown in the photos (here a board with the GPE option installed):





THE OPTIONAL SD MODULE

The optional SD module is used as HD emulation. The module is a common 6 pins microSD module that can be easily found on ebay:



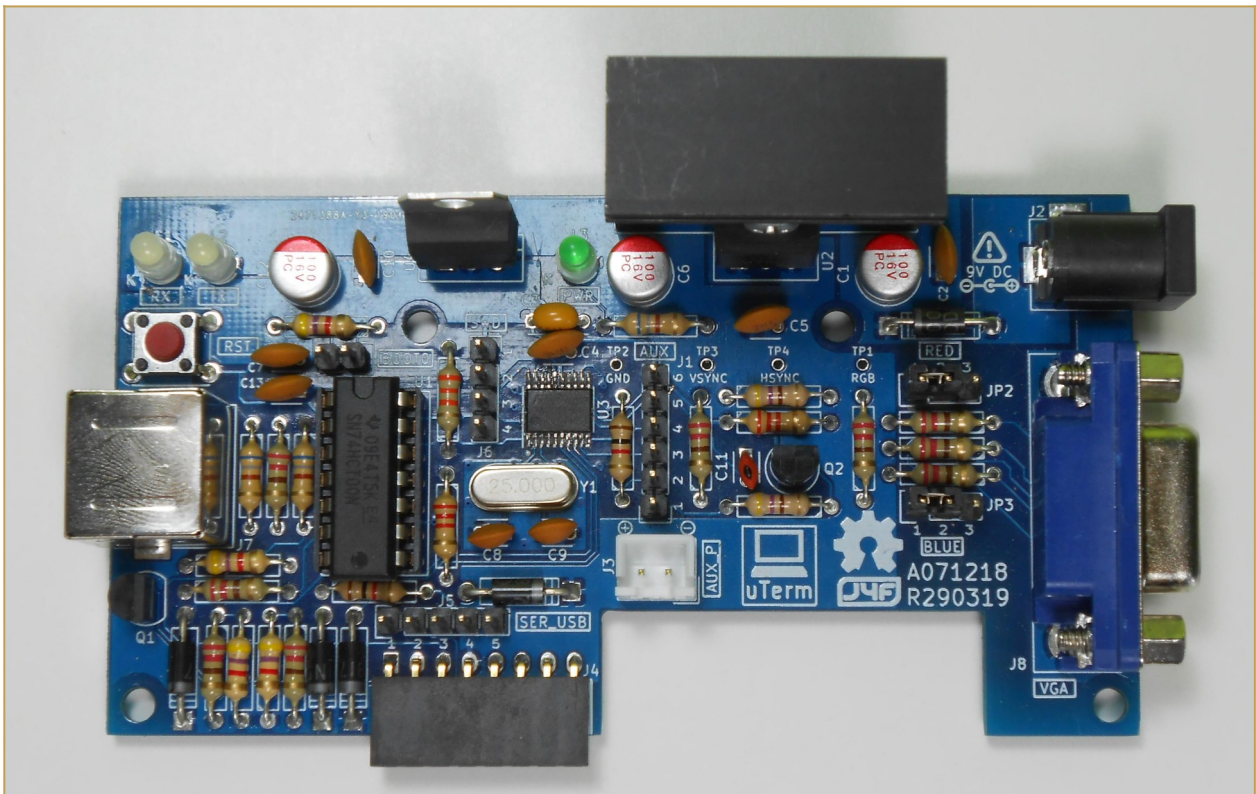
Pay attention on how and where you plug the module in (the only right connector for it is **J6** marked as **SD-MOD**). If you plug it in the wrong connector or in the wrong way it is possible cause permanent damages to both the module and the Z80-MBC2 board!

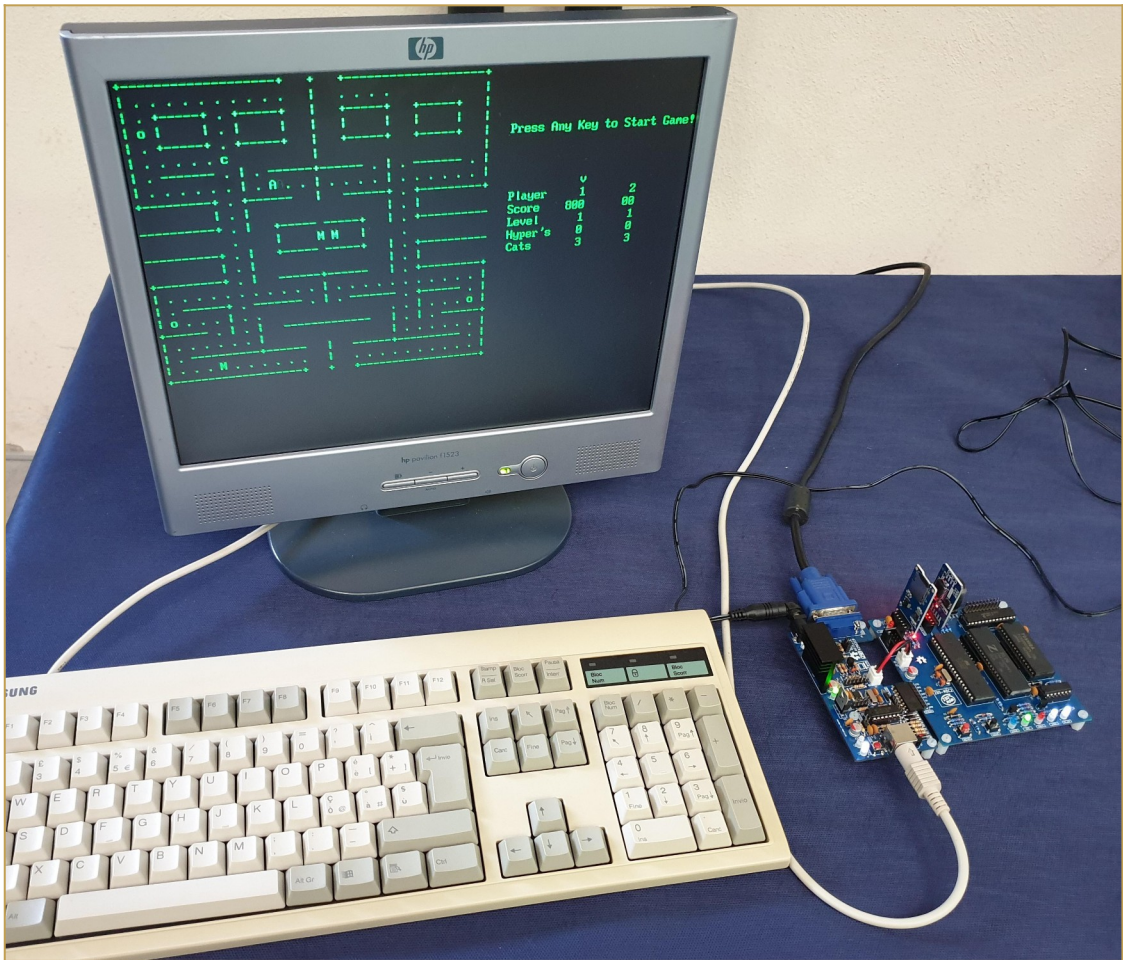
**** uTERM, VT100-LIKE TERMINAL FOR THE Z80-MBC2 ****

uTerm (micro-Term) is a VT100-like terminal for the Z80-MBC2. It has a VGA out and PS/2 keyboard connector, a power supply for the Z80-MBC2 and a **"transparent" serial-USB port**.

uTerm can be mounted **horizontally or vertically** to the Z80-MBC2.

With the uTerm the Z80-MBC2 becomes an "autonomous" computer:



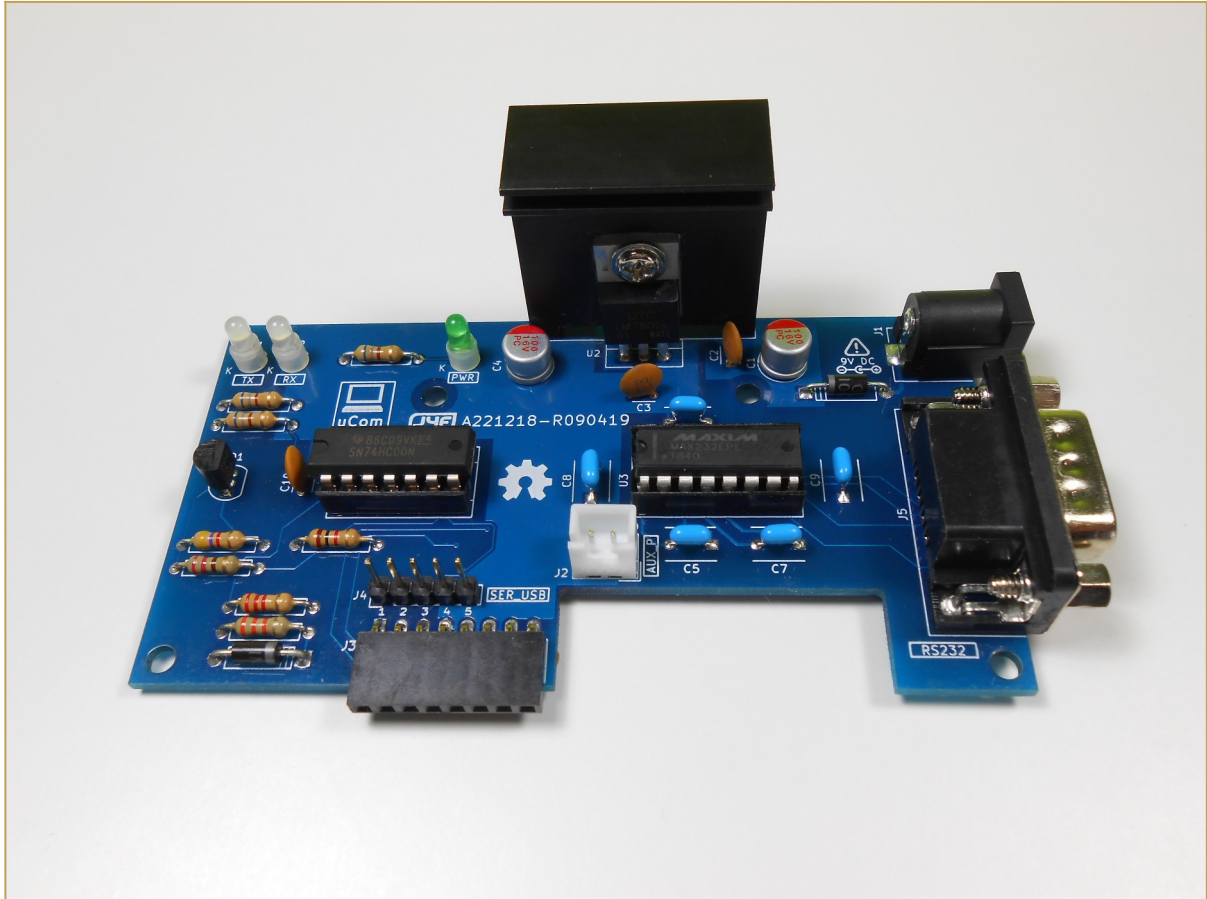


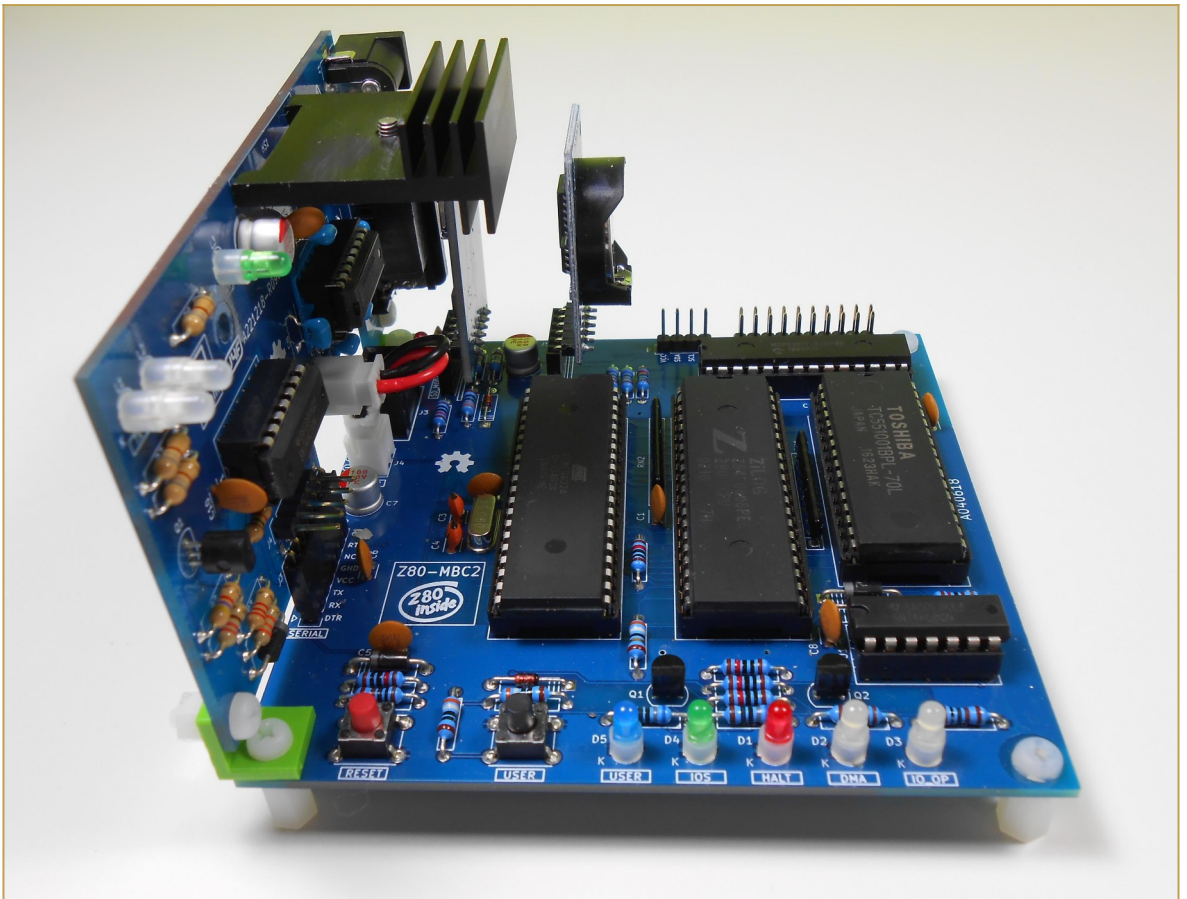
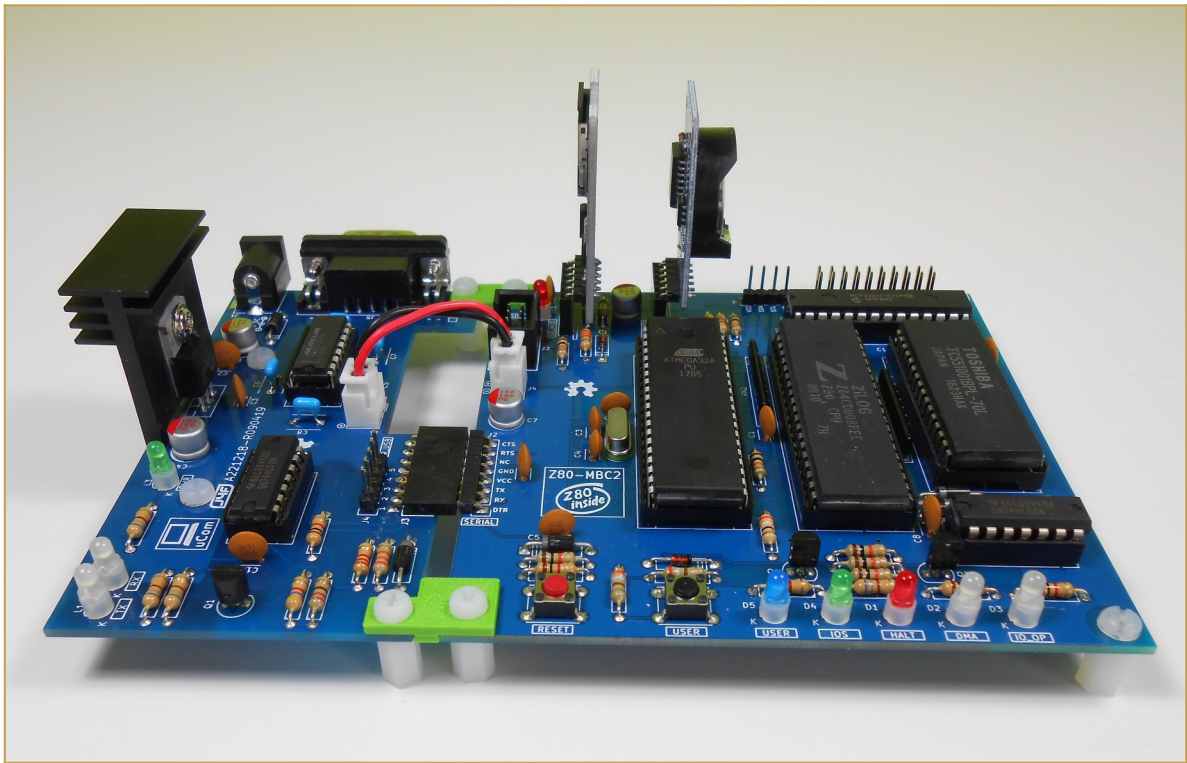
All the details on the uTerm are [here](#).

*** * uCOM, RS232 FOR THE Z80-MBC2 * ***

uCom (micro-Com) is a RS232 adapter for the Z80-MBC2. It has a power supply for the Z80-MBC2 and a **"transparent" serial-USB port**.

uCom can be mounted **horizontally or vertically** to the Z80-MBC2:





With the uCom the Z80-MBC2 can be used with a "vintage" RS232 terminal:



All the details on the uCom are [here](#).

**** SPP (STANDARD PARALLEL PORT) ADAPTER BOARD ****

The **Standard Parallel Port (SPP) Adapter** board allows to use the **GPIO port** of the Z80-MBC2 as a standard printer parallel port.

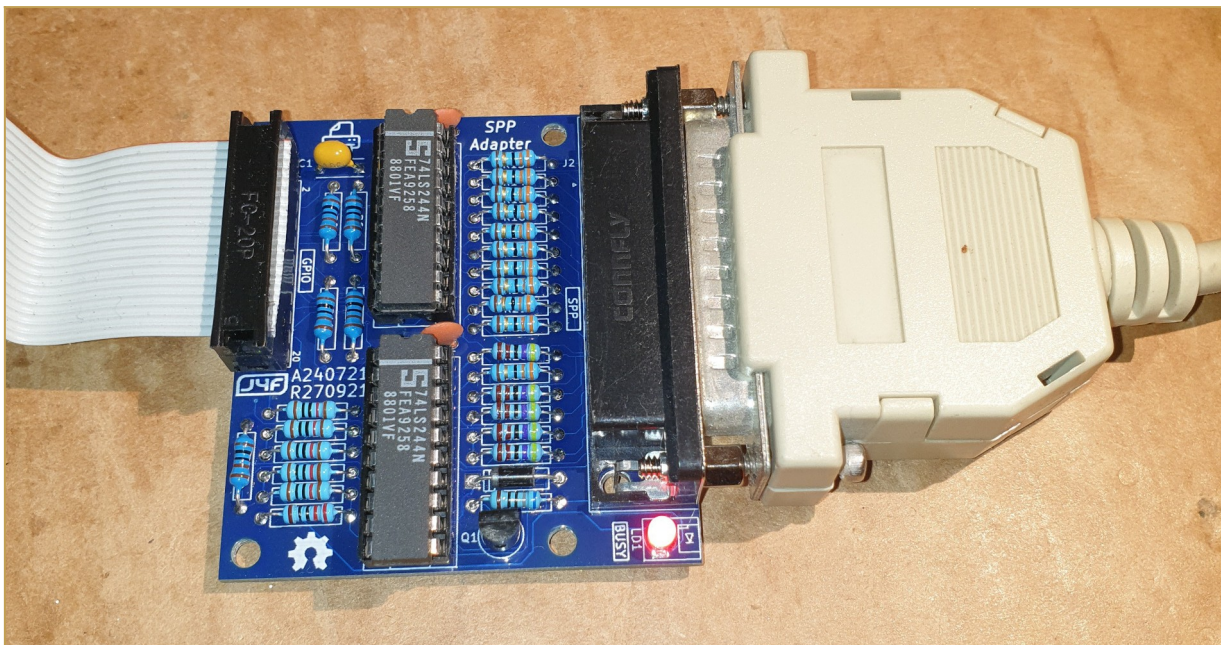
In this way you can use a legacy parallel (Centronics) printer.

To connect the SPP Adapter board to the GPIO connector of the Z80-MBC2 board you need a 10cm long 20 wires flat cable terminated with an IDC connector at both sides (pay attention to connect the cable in the right way on both sides, so the pin 1 on the Z80-MBC2 GPIO connector corresponds to the pin 1 on the SPP Adapter board GPIO connector).

Please note that **you have to power off** the Z80-MBC2 board before connecting or disconnecting the SPP Adapter board to it.

NOTE: before using the **SPP Adapter** board (A240721-R270921, the same board used with the **68K-MBC**) **you have to update** both the IOS firmware and the SD image **to the latest available version** (see the **FILES** section).

In the following image the SPP Adapter board with the flat cable (connected to the GPIO connector of the Z80-MBC2) and with the printer cable:



The cable to use for the printer is the common **parallel printer cable**, with a DB-25 connector at one side and a Centronics connector at the other:



SPP: HOW TO BUILD IT

In the **FILES** section you can find a zip file with all the documentation needed to build the SPP Adapter board, including the Gerber files for the PCB production.

SPP: HOW TO USE THE SPP CP/M UTILITY

To enable the SPP Adapter board under CP/M 2.2 and CP/M3 (banked) I've added on the **drive A:** the custom utility **SPP.BAS**.

You have to execute the **SPP utility** with the command **MBASIC SPP** to enable the SPP Adapter board and "link" to it the **LPT:** CP/M device inside CP/M. After the execution of the SPP utility the GPIO port will be linked and **reserved** (the "normal" GPIO opcodes/functions inside IOS will be disabled) to the SPP parallel port emulation until a system reset or reboot:


```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help

BNKBIOS3 SPR F600 0600
BNKBIOS3 SPR 5300 2D00
RESBDOS3 SPR F000 0600
BNKBDOS3 SPR 2500 2E00

60K TPA

Z80-MBC2 128KB (Banked) CP/M V3.0
Z80-MBC2 BIOS Modules: S200918, S210918-R210923, S220918-R210923, S290918,
S170319

A>mbasic spp
BASIC-80 Rev. 5.21
[CP/M Version]
Copyright 1977, 78, 79, 80 (C) by Microsoft
Created: 15-Dec-80
35835 Bytes free

SPP enabled using the GPIO port.

NOTES:
* GPIO port is now reserved exclusively for SPP emulation
* The SPP emulation is active until next system reset/reboot
* When SPP is active a permanent not ready printer (e.g. printer off or
not connected) can hang CP/M

A>|
```

NOTE: the SPP Adapter board is currently supported under CP/M 2.2 and CP/M 3 (banked) only.

SPP: WHERE TO GET A PCB

I've prepared an "easy" link to get a small lot (5 pcs minimum) of PCB of the *SPP* Adapter. The link is [this one](#).

SPP: HOW TO GET A KIT OR AN ASSEMBLED UNIT

If you are looking for a SPP Adapter board kit with all the needed parts or an assembled unit ready to use now there is a professional seller that can sell both and ship worldwide.

The link to the seller is [this one](#).

* * SOFTWARE OVERVIEW * *

The MCU Atmega32A is used as universal I/O subsystem, as Eeprom, and as reset and clock generator for the Z80 CPU.

The software running into the Atmega32A is the **IOS** (*Input Output Subsystem*) written using the **Arduino IDE** environment.

The IOS allows to interface the Atmega32A directly with the CPU system bus, emulating the needed I/O chips during the *I/O read*, *I/O write* and *IRQ acknowledge* CPU bus cycles (see the Z80 datasheet).

Furthermore, the IOS loads the RAM during the boot phase, "feeding" the CPU with the necessary instructions.

It is possible to choose between two different "flavors" of IOS: **IOS** and **IOS LITE**:

| | SD Module | RTC Module | GPE | Embedded iLoad | Embedded Basic/Forth | Default serial speed |
|----------|-----------|------------|-----------|----------------|----------------------|----------------------|
| IOS | Supported | Supported | Supported | Supported | on SD | 115200 (8N1) |
| IOS LITE | - | Supported | Supported | Supported | Supported | 9600 (8N1) |

IOS LITE is more intended for testing the board for the base functions (it doesn't support the SD module, so only iLoad, the embedded stand-alone Basic and the embedded stand-alone Forth can be used). For normal use the standard IOS is the one to flash.

I've "ported" the stand-alone Basic interpreter to the Z80-MBC2 using the sources provided in the great Grant Searle **site** , after the needed modification due the different HW design (in the Grant's site is requested an acknowledgement to his site to use this source, so I did and I have also emailed to him about this thing).

The resulting ROM image is stored inside the Atmega32A (only for IOS-LITE) and loaded in the TC551001 RAM by the Atmega32A during the system boot. The original manual of this Basic interpreter is **here**.

```
COM3 - Z80-MBC Terminal VT
File Edit Setup Control Window Help

Z80-MBC2 - A040618
IOS-LITE - I/O Subsystem - S220618

IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (25/07/18 12:54:04)
IOS: RTC DS3231 temperature sensor: 27C
IOS: Found GPE Option
IOS: Loading boot program... Done
IOS: Z80 is running from now

uBIOS - S210718 (Adapted from Z80 SBC by G. Searle)

Memory top?
Z80 BASIC Ver 4.7b
Copyright (C) 1978 by Microsoft
56958 Bytes free
Ok
```

The Forth stand-alone interpreter is a modified version (for the Z80-MBC2) of the one provided by **Bill Westfield** for the Z80-MBC.

```
COM3 - Z80-MBC Terminal VT
File Edit Setup Control Window Help

Z80-MBC2 - A040618
IOS-LITE - I/O Subsystem - S220618

IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (25/07/18 13:00:09)
IOS: RTC DS3231 temperature sensor: 27C
IOS: Found GPE Option
IOS: Loading boot program... Done
IOS: Z80 is running from now

8080 fig-FORTH 1.3
( ***** OK
(          OK
( Blink test - Forth - Z80-MBC2 OK
( OK
( To execute give the command "blink" OK
( OK
( ***** OK
: opcd 0 1 P! ; OK
: ledon 1 0 P! ; OK
: ledoff 0 0 P! ; OK
: delay 8000 0 D0 NOOP LOOP ; OK
: blink CR ." Blinking..." CR BEGIN opcd ledon delay opcd ledoff delay 0 UNTIL ;OK
blink
Blinking...
```

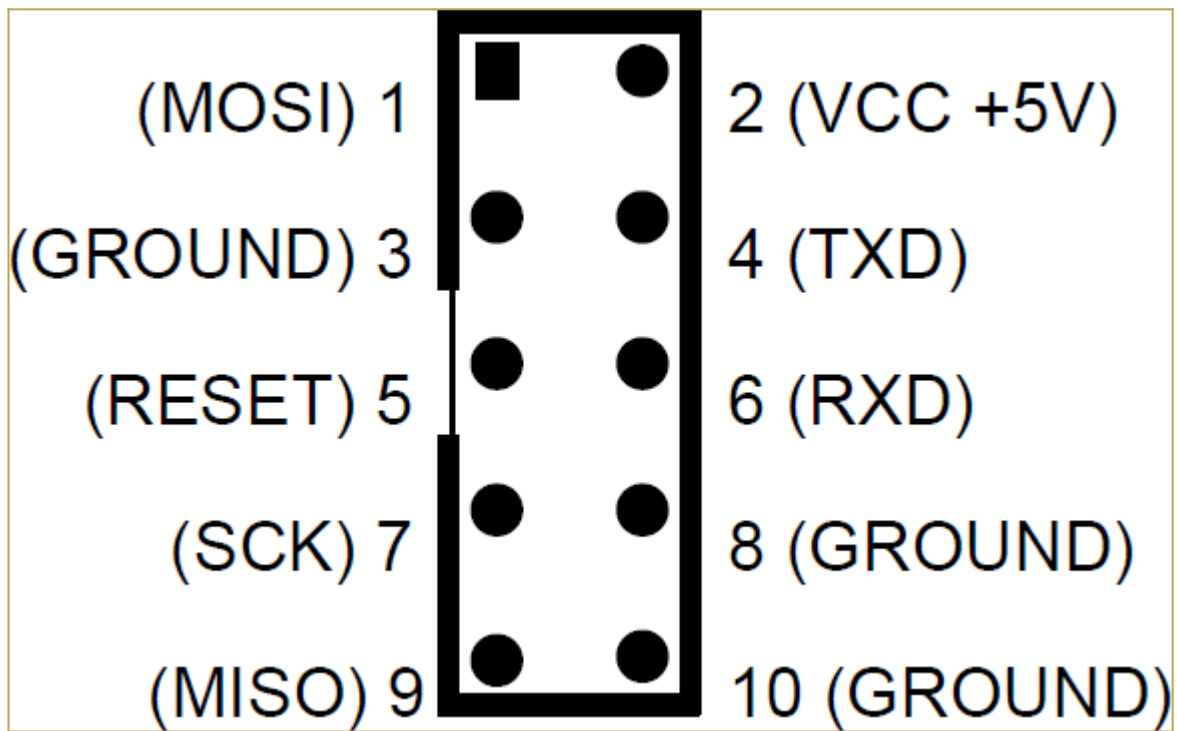

HOW FLASH THE BOOTLOADER FROM ARDUINO IDE (LINUX)

A cheap and easy way to burn the Arduino bootloader is to use an *USBasp programmer* that is commonly available:



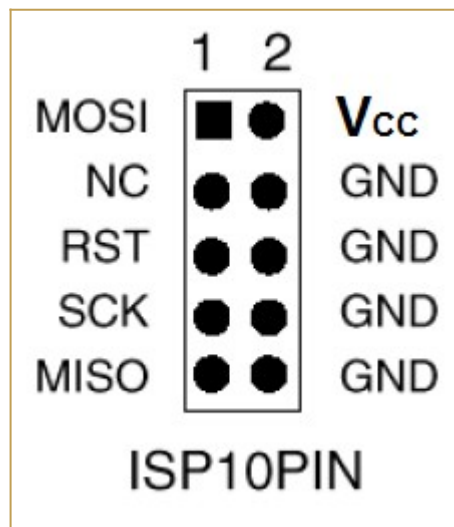
The USBasp is also capable to give the power to the "target" using the VCC pin, but remember to **check that the JP1 jumper is set to provide 5V to the target** (as shown in the photo).

Please note that **the pinout of the USBasp is a little different from the "standard" ICSP (or ISP) pinout:**

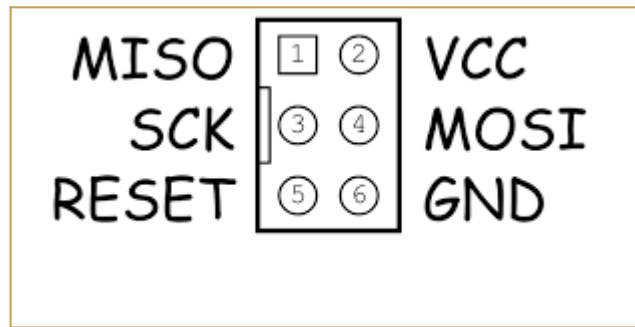


In the previous picture it is possible to see that pins 4 (TXD) and 6 (RXD) are not at GND as expected by the standard ICSP port, and pin 3 is not NC.

See the following picture showing the standard 10 pin ICSP pinout:



So you must consider this when connecting the USBasp to the **6 pins ICSP port (J3)** on the Z80-MBC2 (see the schematic):

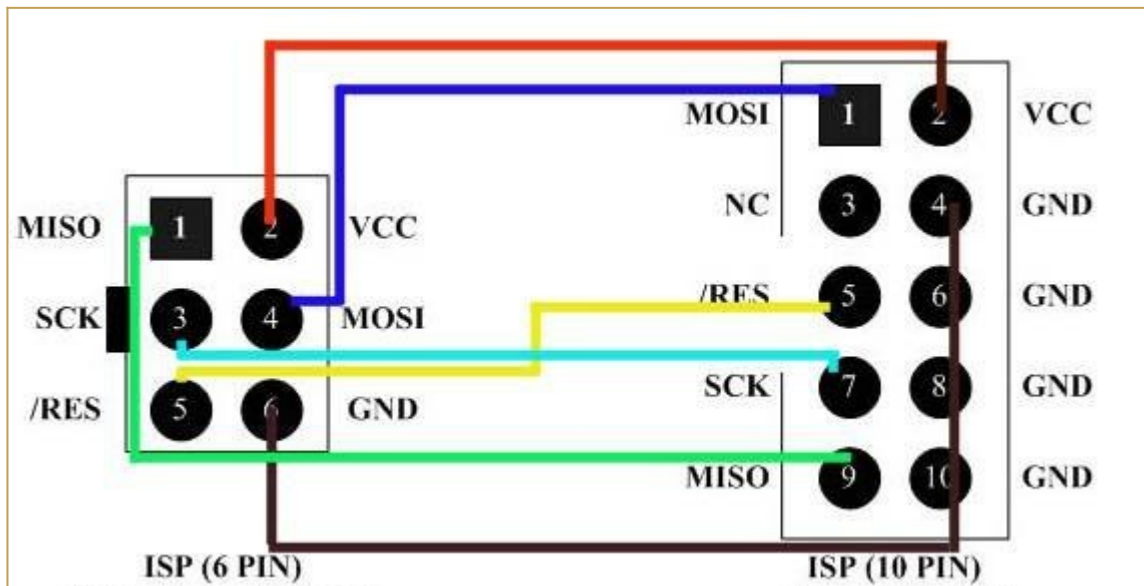


To avoid problems I suggest **to use as GND pin 10 of the USBasp connector**, and connect the other pins (VCC, MISO, MOSI, SCK, RST) accordingly. An handy way to connect the USBasp to the 6 pin ICSP port (J3) of the Z80-MBC2 **could be** to use a commonly available "10pin to 6pin" adapter like this:

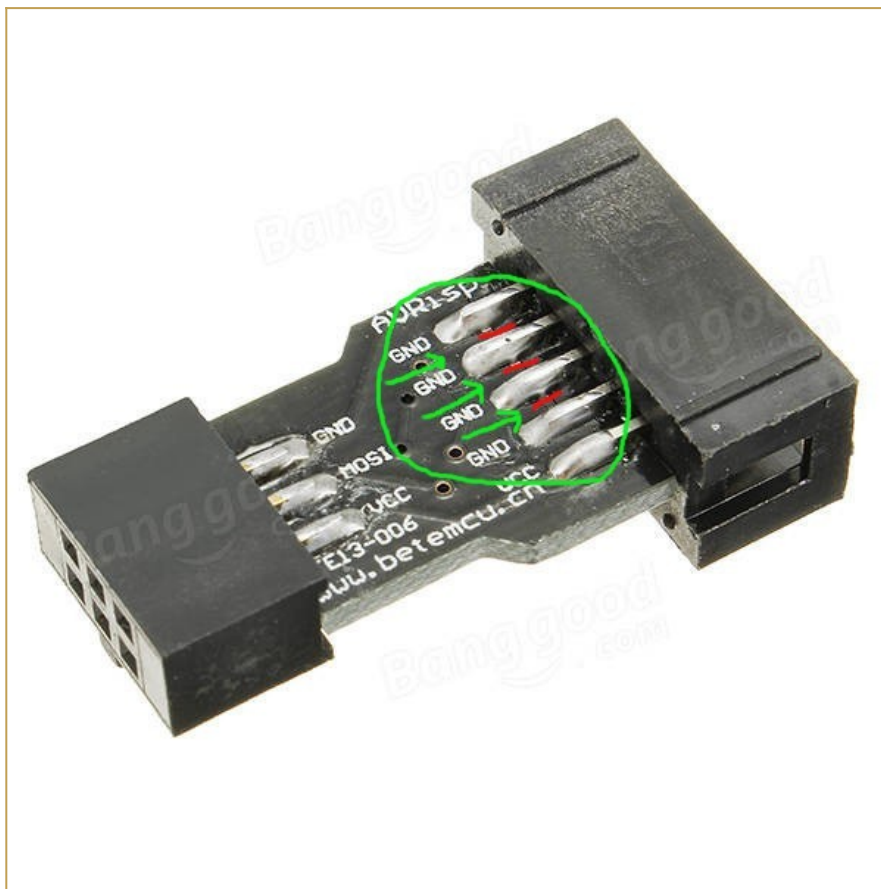


but I suggest not to use it "as is" because its internal connections are done for a "standard" ICSP port, and we have seen that the USBasp connector differs from the

standard one. The schematic of the adapter shows that isn't compatible "as is" with the UABasp connector:



To use it is a good idea isolate the pins 4, 5 and 6 cutting the trace on the PCB of the adapter that connects those pins together, and then check with a tester. In the following photo are shown the three cuts (thin red lines inside the green "circle") to do:



To easily burn the bootloader from Arduino IDE follow these "**quick and dirty**" steps (tested on a linux Mint OS with Arduino IDE 1.8.19):

STEP 1: Connect the 10 pins connector of the USBasp programmer to the 6 pins ICSP port (J3) of the Z80-MBC2 (using wires or a modified adapter as discussed before);

STEP 2: Verify carefully that **any other connector of the Z80-MBC2 is not used**, and verify that **both the SD and RTC modules (if present) are removed** from the board;.

STEP 3: Only at this point connect the USB side of the USBasp programmer to an USB port of your workstation;

STEP 4: Open a "terminal" window on your workstation and **go to the directory where there are the Arduino IDE executables**, and get the root privileges with the command:

```
sudo su
```

then run the Arduino IDE with the command:

```
./arduino
```

STEP 5: Because Arduino IDE is running as the root user it is necessary re-install the "core" for the Atmega32. Open the Board Manager as you already did (anyway the guide is **here**). Note that you must do this step only the first time you execute the Arduino IDE as root;

STEP 6: Now from the **Tools** menu of Arduino IDE select "**Atmega32**" as "**Board**", "**16 MHz external**" as "**Clock**", and "**USBasp**" as "**Programmer**". Then you can burn the right bootloader (without playing with the FUSE setting) selecting "**Burn Bootloader**" from the same "**Tools**" menu.

All done!

NOTE: If you use a different method requiring manual settings, the right **Fuse bits** setting to use is: **High Byte 0xD6, Low Byte 0xAF, Lock Byte 0xCF**.

HOW ENTER IN THE "SELECT BOOT MODE OR SYSTEM PARAMETERS" MENU

To enter in the "Select boot mode or system parameters" menu (or simply "boot menu") you must press the **RESET key** (SW2), release it and press immediately the **USER key** (SW1) and keep it pressed until the IOS led starts to blink.

An other way is to press both keys, release the **RESET key** holding the **USER key** down until the IOS led starts to blink, or you see the menu on the screen.

In the following screenshots is shown the menu when both the RTC module and the GPE option are installed for IOS-LITE and IOS:

```
COM3 - Z80-MBC Terminal VT
File Edit Setup Control Window Help

Z80-MBC2 - A040618
IOS-LITE - I/O Subsystem - S220618

IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (25/07/18 12:55:51)
IOS: RTC DS3231 temperature sensor: 27C
IOS: Found GPE Option

IOS: Select boot mode or system parameters:

0: No change (->1)
1: Basic
2: Forth
3: iLoad
4: Change Z80 clock speed (4/8MHz)
5: Change RTC time/date

Enter your choice >
```

```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help

Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R290823

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (24/09/23 16:50:00)
IOS: RTC DS3231 temperature sensor: 28C
IOS: CP/M Autoexec is OFF

IOS: Select boot mode or system parameters:

0: No change (3)
1: Basic
2: Forth
3: Load/set OS Disk Set 0 (CP/M 2.2)
4: Autoboot
5: iLoad
6: Change Z80 clock speed (->4MHz)
7: Toggle CP/M Autoexec (->ON)
8: Set serial port speed (115200)
9: Set RTC time/date

Enter your choice >|
```

iLoad: loads and executes a Z80 Intel-Hex formatted executable sent from the serial port;

Autoboot: loads and executes a Z80 binary file (AUTOBOOT.BIN) on SD;

Load/set OS Disk Set <n>: loads or changes and runs an Operative System installed into the Disk Set <n> on SD;

Toggle CP/M Autoexec: Turns on or off the execution of the AUTOEXEC batch file at the

cold boot. This is supported for CP/M 2.2, CP/M 2.71 and CP/M 3.

The remaining choices are self-explanatory.

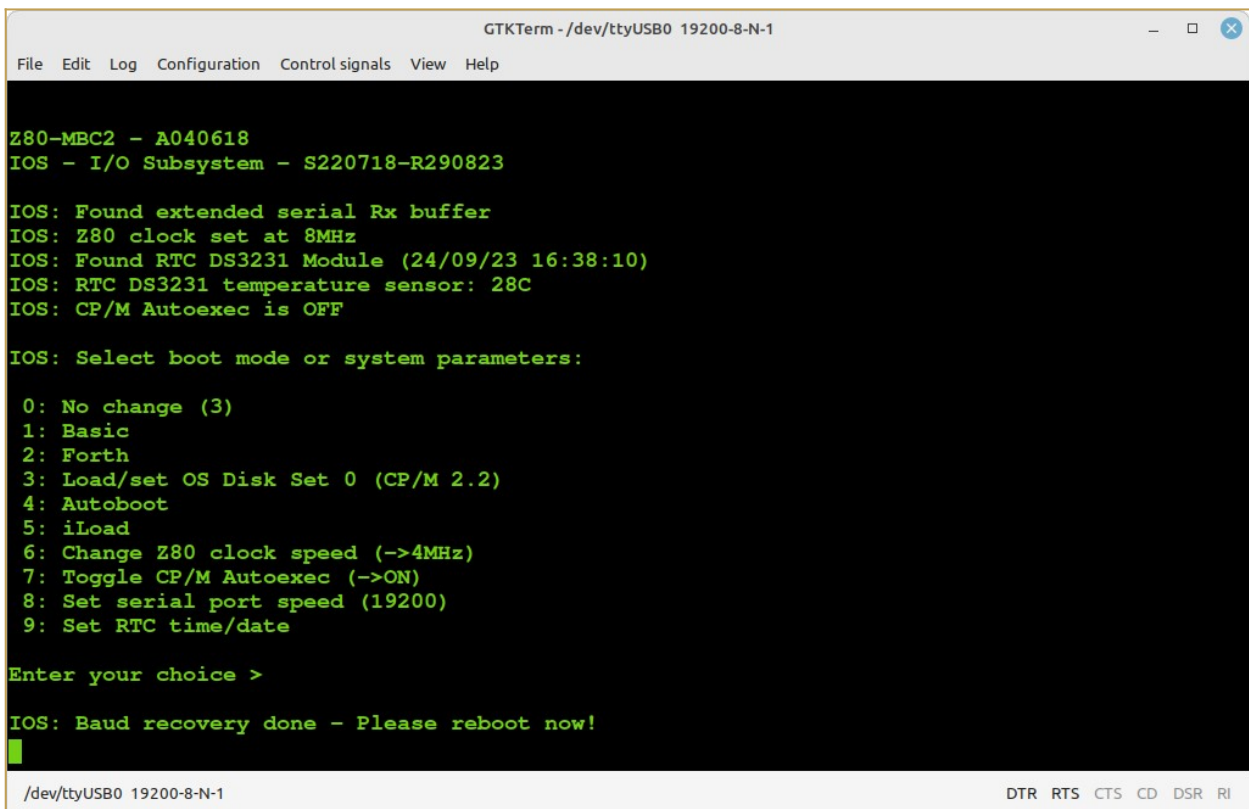
In the following it will be assumed the use of the standard IOS, as the IOS-LITE is intended only for a limited use.

THE BAUD RECOVERY VIRTUAL BUTTON

Changing the speed of the serial port may happen "to loose" the control of the board (i.e. you forget the speed or set a wrong speed). In this case you can use a "virtual button" to reset the serial port to the default speed (115200 bps) without the need of a terminal.

To activate the Baud recovery "virtual button" you have to press both the **RESET** and **USER** key, release the **RESET** key holding the **USER** key down until the **USER** led starts to blink (like for the "Select boot mode or system parameters" menu) and keeping it down at least for 4 seconds more until both the **USER** and **IOS** led start to blink **very quickly**. This is the sign that the Baud Recovery "virtual button" has been activated.

At the next reboot the serial port will be set at the default speed (115200 bps):



```
GTKTerm - /dev/ttyUSB0 19200-8-N-1
File Edit Log Configuration Control signals View Help

Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R290823

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (24/09/23 16:38:10)
IOS: RTC DS3231 temperature sensor: 28C
IOS: CP/M Autoexec is OFF

IOS: Select boot mode or system parameters:

0: No change (3)
1: Basic
2: Forth
3: Load/set OS Disk Set 0 (CP/M 2.2)
4: Autoboot
5: iLoad
6: Change Z80 clock speed (->4MHz)
7: Toggle CP/M Autoexec (->ON)
8: Set serial port speed (19200)
9: Set RTC time/date

Enter your choice >

IOS: Baud recovery done - Please reboot now!
```

Note: The Baud Recovery virtual button can be triggered only if the serial port is set to a non-default (115200) value.

THE SD IMAGE

The content of the microSD (I'll call it simply SD from now) is compressed into a zip file in the **Files** section.

When you update the IOS firmware you must always update the content of the SD too, as the SD image is normally suited for a given IOS revision.

You have to unzip it retaining the structure of the sub-directories into a FAT formatted SD card, so that the various root files (inside the .zip as the various .DSK files and so on...) are in the root of the SD itself.

IOS supports only **FAT16** and **FAT32**. A 1GB SD is more than enough, anyway because they tends to be difficult to find now a 4GB SD can be a good choice.

About the SD technology, only "legacy" SD (aka SDSC with a capacity up to 2GB) and SDHC cards (2GB - 32GB) can be used. Other most recent types are not supported (so, no SDXC, SDUC,...).

What it really needed to let IOS run are only all the files in the root folder. The other sub-directories contain source files or examples or other kind of content.

Inside every sub-directory there is a **README.TXT** file that *may contain important info/updates*. Please read them all when you use a SD image first time or when update it!

In the root there is a **ChangeLog.txt** file with the changes log (related to the SD image content).

HOW ADD CP/M FILES INSIDE A VIRTUAL DISK USING CPMTOOLSGUI

The Z80-MBC2 maps any disk like A: B: C: etc. into an image file on SD card with this file name: **DSxNyy.DSK**;

where **x** (from 0 to 9) is OS:

0 = CP/M 2.2

1 = QP/M 2.71

2 = CP/M 3

.....

and **yy** (from 00 to 15) is the disk (00 = A: 01 = B: etc.).

You can download **CpmtoolsGUI** (English Windows version) from **here**.

Extract the file CpmtoolsGUI.exe in a new folder and add/overwrite the file **diskdefs** copying it from the folder **cpmtools** inside the SD.

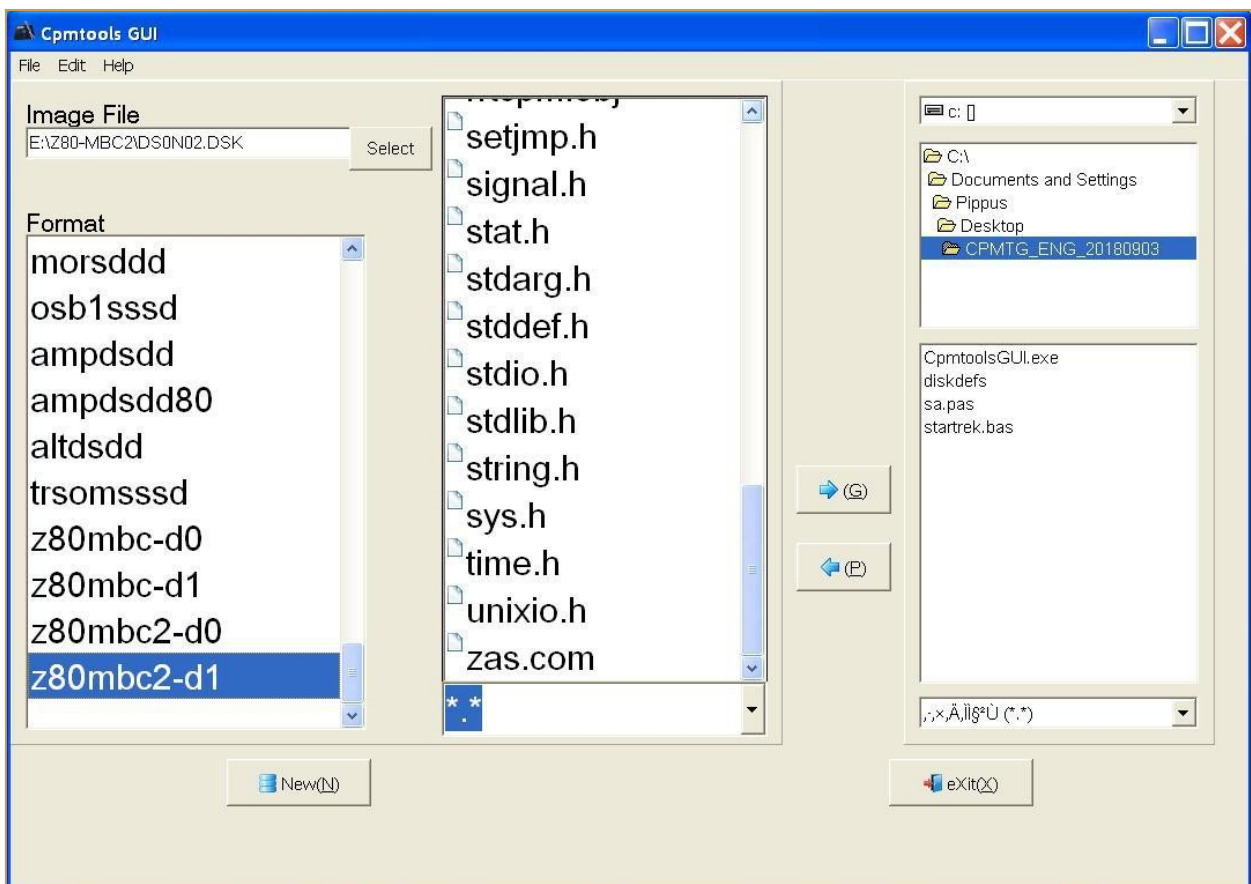
-> STEP 1

Select in the upper left window (Image File) of the CpmtoolsGUI tool the virtual disk where you want to add files.

For CP/M 2.2 and QP/M 2.71:

select "**z80mbc2-d0**" only for disk 0 or "**z80mbc2-d1**" for the others (disk 1 - 15) in the bottom left window (Format) of CpmtoolsGUI.

In the following image is selected (Image File) the disk **DS0N02.DSK** that corresponds to the **disk C:** (yy = disk = 02) of the **CP/M 2.2** OS (x = 0):



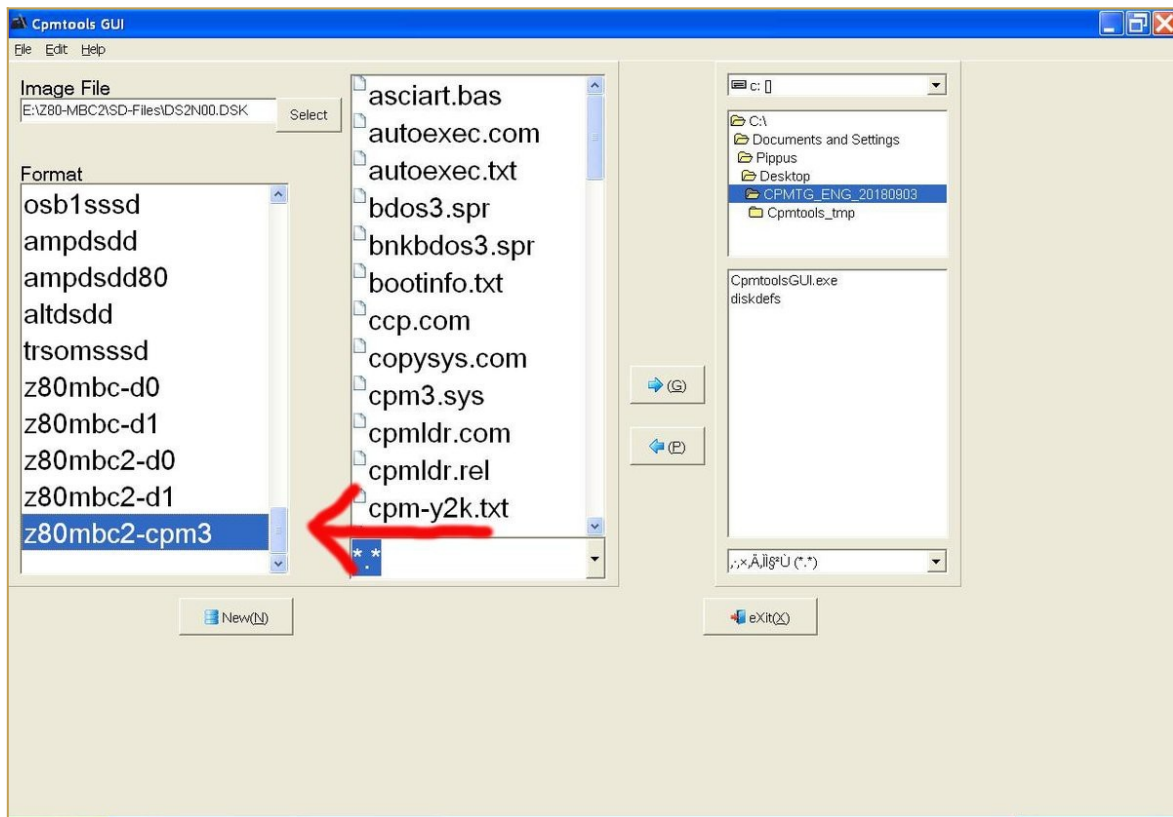
In the center window you can see all the files inside the selected virtual disk (disk 0 - 15).

Please note that if you choose an empty disk (like P:) you won't see any file name in the center window of the CpmtoolsGUI tool.

For CP/M 3:

select in the bottom left window (Format) of CpmttoolsGUI "z80mbc2-cpm3" for any disk.

In the following image is selected (Image File) the disk **DS2N00.DSK** that corresponds to the **disk A:** (yy = disk = 00) of the **CP/M 3** OS (x = 2):



-> STEP 2

To add one or more files to the selected virtual disk you have simply point the upper right selection window to the folder where the new files are stored in your PC, select them using the bottom right selection window and press the "<P" button. After the add you'll see the added file names in the center window (together with the others file previously present).

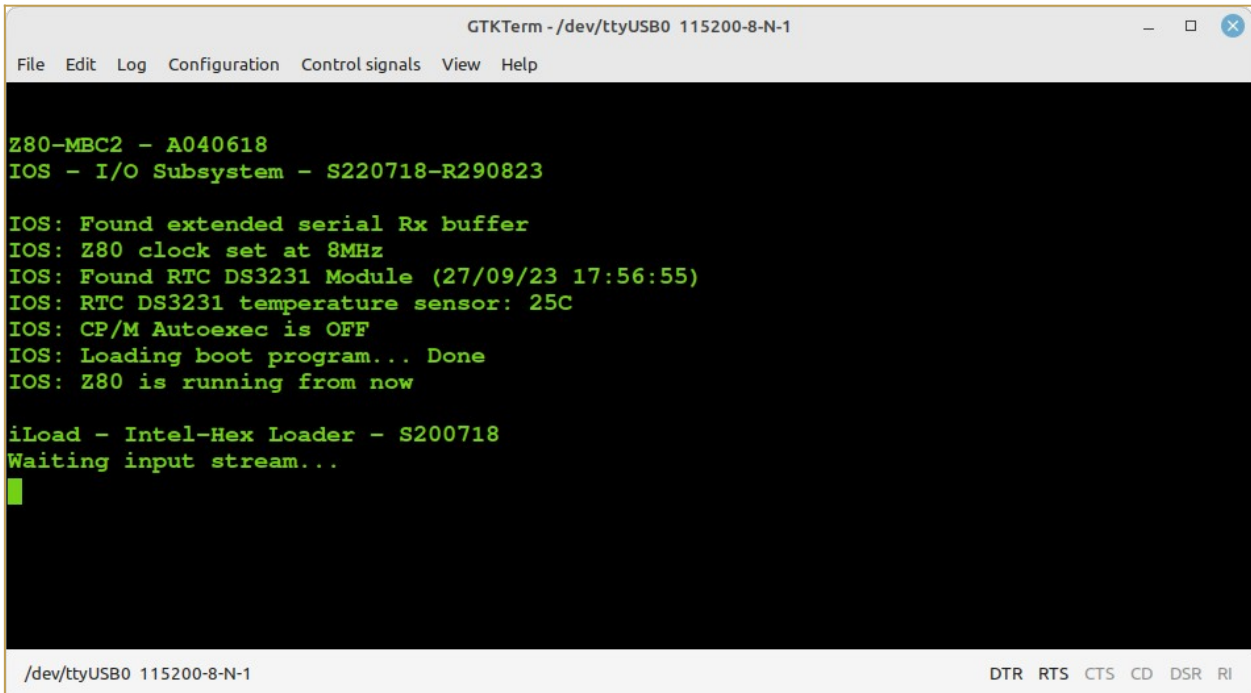
-> STEP 3

Exit from the the CpmttoolsGUI tool pressing the **eXit** button.

HOW TO USE ILOAD

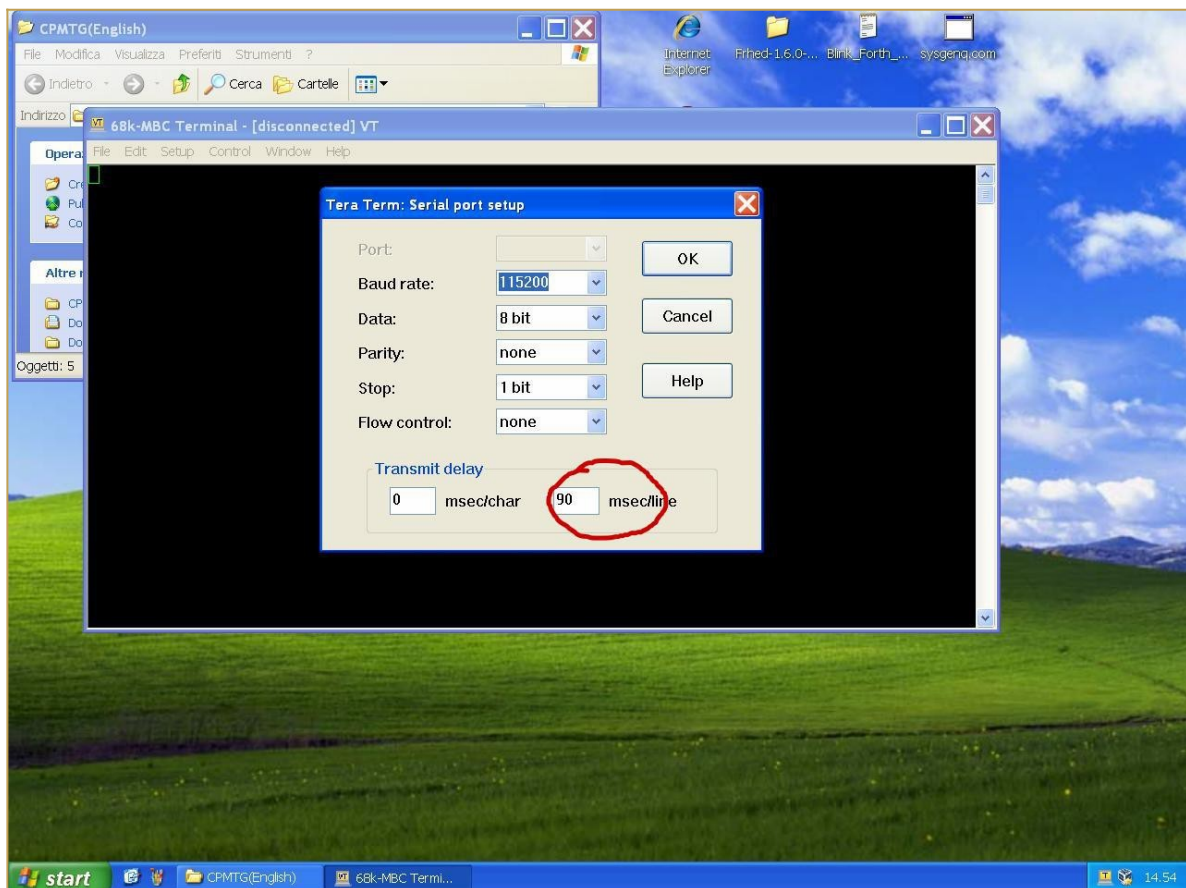
iLoad is an Intel-Hex format loader that allows to load from the serial port a binary program using the Intel-Hex format, and execute it. You can run it selecting the option 5

from the boot menu (with IOS-LITE the option is 3):



When using *iLoad* remember to set a **50/90ms delay** on every transmitted line into the serial port setting of the SW terminal you are using.

In the following image there is the setting window for TeraTerm inside a Windows XP VM:



This is required because the standard serial port of the Arduino firmware doesn't use any handshaking control.

Using the iLoad boot mode it's possible to automate all the process from the source to the execution in the target.

Please remember that iLoad will take **the first address** of the Intel-Hex stream **as the starting address of the program**, and after the loading will jump to it.

iLoad will also check the hex stream for errors, and protects itself if "someone" try to load a program (or a part) over itself ("**illegal address**" error).

HOW ENABLE THE EXTENDED RX BUFFER FOR XMODEM (CP/M)

Because the Z80-MBC2 uses a virtual serial port without handshaking there is a timing problem when dealing with the 128 bytes packets used by the XMODEM protocol.

So the support to the XMODEM protocol has requested changes to extend the serial port RX buffer to 128 bytes.

Thanks to user **Hans** who pointed me to the right direction, there is a simple way to modify the size of the RX buffer used for the serial port.

Search the file **boards.txt** related with the MightyCore variant in your Arduino IDE.

In a typical Linux Arduino IDE installation it is located in the hidden directory:

/home/<username>/.arduino15/packages/MightyCore/hardware/avr/2.0.5/

or in a Windows 10 installation:

C:\Users\<username>\AppData\Local\Arduino15\packages\MightyCore\hardware\avr\2.0.5

Open board.txt with an editor and locate the section related to the Atmega32/A:

The screenshot shows a text editor window titled 'boards.txt - Kate'. The menu bar includes File, Edit, View, Projects, Bookmarks, Sessions, Tools, Settings, and Help. The toolbar has icons for Open, New, Save, Save As, Close, Undo, and Redo. The main text area shows the following code:

```
602 164.menu.clock.1MHz_internal.build.f_cpu=1000000L
603
604
605
606 #####
607 #### ATmega32/A ####
608 #####
609
610 # General
611 32.name=ATmega32
612 32.upload.tool=avrdude
613 32.upload.maximum_data_size=2048
614 32.bootloader.tool=avrdude
615 32.bootloader.unlock_bits=0x3f
616 32.bootloader.lock_bits=0x0f
617 32.bootloader.low_fuses=0b{bootloader.bod_bits}{bootloader.sut_cksels_bits}
618 32.bootloader.high_fuses=0b110{bootloader.ckopt_bit}011{bootloader.bootrst_bit}
619 32.build.core=MCUdude_corefiles
620 32.build.board=AVR_ATmega32
621 32.build.mcu=atmega32
622
623 # Upload port select
624 32.menu.bootloader.uart0=Yes (UART0)
625 32.menu.bootloader.uart0.upload.maximum_size=32256
```

The status bar at the bottom indicates 'Line 749, Column 23' and 'INSERT Soft Tabs: 4 UTF-8 Normal'.

go some lines down until you see the "**32.menu.LTO.Os.compiler.cpp.extra_flags=**" line:

The screenshot shows the same text editor window, now scrolled down to lines 641-664. The code is as follows:

```
641 32.menu.pinout.bobuino.build.variant=bobuino
642 32.menu.pinout.bobuino.build.bootloader_led=B7
643 32.menu.pinout.sanguino=Sanguino pinout
644 32.menu.pinout.sanguino.build.variant=sanguino
645 32.menu.pinout.sanguino.build.bootloader_led=B0
646
647 # Brown out detection - This is the first part of the low fuse bit concatenation
648 32.menu.BOD.2v7=BOD 2.7V
649 32.menu.BOD.2v7.bootloader.bod_bits=10
650 32.menu.BOD.4v0=BOD 4.0V
651 32.menu.BOD.4v0.bootloader.bod_bits=00
652 32.menu.BOD.disabled=BOD disabled
653 32.menu.BOD.disabled.bootloader.bod_bits=11
654
655 # Compiler link time optimization
656 32.menu.LTO.Os=LTO disabled
657 32.menu.LTO.Os.compiler.c.extra_flags=
658 32.menu.LTO.Os.compiler.c.elf.extra_flags=
659 32.menu.LTO.Os.compiler.cpp.extra_flags=
660 32.menu.LTO.Os.ltoarcmd=avr-ar
661
662 32.menu.LTO.Os_flto=LTO enabled
663 32.menu.LTO.Os_flto.compiler.c.extra_flags=-Wextra -flto -g
664 32.menu.LTO.Os_flto.compiler.c.elf.extra_flags=-w -flto -g
```

Line 659 is highlighted in blue. The status bar at the bottom indicates 'Line 659, Column 1' and 'INSERT Soft Tabs: 4 UTF-8 Normal'.

then append the string "**-D SERIAL_RX_BUFFER_SIZE=128**" to that line:

```
boards.txt
647 # Brown out detection - This is the first part of the low fuse bit concatenation
648 32.menu.BOD.2v7=BOD 2.7V
649 32.menu.BOD.2v7.bootloader.bod_bits=10
650 32.menu.BOD.4v0=BOD 4.0V
651 32.menu.BOD.4v0.bootloader.bod_bits=00
652 32.menu.BOD.disabled=BOD disabled
653 32.menu.BOD.disabled.bootloader.bod_bits=11
654
655 # Compiler link time optimization
656 32.menu.LTO.Os=LTO disabled
657 32.menu.LTO.Os.compiler.c.extra_flags=
658 32.menu.LTO.Os.compiler.c.elf.extra_flags=
659 32.menu.LTO.Os.compiler.cpp.extra_flags=-DSERIAL_RX_BUFFER_SIZE=128
660 32.menu.LTO.Os.ltoarcmd=avr-ar
661
662 32.menu.LTO.Os_flto=LTO enabled
663 32.menu.LTO.Os_flto.compiler.c.extra_flags=-Wextra -flto -g
664 32.menu.LTO.Os_flto.compiler.c.elf.extra_flags=-w -flto -g
665 32.menu.LTO.Os_flto.compiler.cpp.extra_flags=-Wextra -flto -g
666 32.menu.LTO.Os_flto.ltoarcmd=avr-gcc-ar
667
668 # Clock frequencies - This is the second part of the low fuse bit concatenation
669 32.menu.clock.16MHz_external=External 16 MHz
670 32.menu.clock.16MHz_external.upload.speed=115200
671 32.menu.clock.16MHz_external.bootloader.extra_flags=1111111
```

Save the edited board.txt file.

All done!

At this point you can recompile **with the LTO option disabled** and flash the IOS inside the Atmega32A with the extended RX buffer enabled.

Please remember that if you update the MightyCore you will lose the changes. In this case re-apply the previous steps.

Note that IOS checks if this extended buffer is active, and in this case will print a status line during the boot phase ("**IOS: Found extended serial Rx buffer**").

CP/M 2.2

To run CP/M 2.2 select it from the boot menu setting the Disk Set 0:

```

GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
IOS - I/O Subsystem - S220718-R290823

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (27/09/23 17:58:52)
IOS: RTC DS3231 temperature sensor: 26C
IOS: CP/M Autoexec is OFF
IOS: Current Disk Set 0 (CP/M 2.2)
IOS: Loading boot program (CPM22.BIN)... Done
IOS: Z80 is running from now

Z80-MBC2 CP/M 2.2 BIOS - S030818-R120923
CP/M 2.2 Copyright 1979 (c) by Digital Research

A>d
ASCIART  BAS  4k : ASM      COM  8k : AUTOEXEC SUB  4k : AUTOEXEC TXT  4k
CATCHUM  COM 32k : CATCHUM DAT  4k : CATCONF COM 24k : D      COM  4k
DDT      COM  8k : DUMP    COM  4k : ED      COM  8k : GENHEX  COM  4k
GORILLA  COM 32k : GPELED  BAS  4k : GPIO   BAS  4k : HELLO  ASM  4k
HELLO    COM  4k : LADCONF COM 24k : LADDER  COM 40k : LADDER  DAT  4k
LOAD     COM  4k : MAC     COM 16k : MBASIC  COM 24k : MBASIC85 COM 24k
PEG      COM  8k : PIP     COM  8k : RTC    BAS  4k : SHOWWARM BAS  4k
SPP      BAS  4k : STARTREK BAS 24k : STAT   COM  8k : SUBMIT  COM  4k
TREKINST BAS  8k : USERLED BAS  4k : XMODEM CFG  8k : XMODEM  COM  8k
XSUB     COM  4k : ZDE16   COM 20k : ZDENST16 COM 12k
A: Total of 420k in 39 files with 7740k space remaining.
A>

```

To add, extract or delete files inside a virtual disk (virtual disks filenames on SD are "DS0Nxx.DSK", where "xx" is the disk number) see the paragraph: **HOW ADD CP/M FILES INSIDE A VIRTUAL DISK USING CPMTOOLSGUI.**

NOTE: The creation of a new CP/M 2.2 boot disk (the first disk DS0N00.DSK) requires further processing (track 0 handling), so is recommended only to add, extract or delete files inside the boot virtual disk (A:).

CP/M 2.2 WARM BOOT MESSAGE

Starting with IOS S220718-R290823 the message shown when CP/M 2.2 makes a warm boot is no more present.

If you are curious about how CP/M 2.2 handles this event it is possible to re-activate it easily.

I've added in the **drive A:** the utility **SHOWWARM.BAS** to enable the "warm boot" message.

To execute give the command: **MBASIC SHOWWARM:**


```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
IOS: Found GPE Option
IOS: CP/M Autoexec is OFF
IOS: Current Disk Set 0 (CP/M 2.2)
IOS: Loading boot program (CPM22.BIN)... Done
IOS: Z80 is running from now

Z80-MBC2 CP/M 2.2 BIOS - S030818-R120923
CP/M 2.2 Copyright 1979 (c) by Digital Research

A>mbasic showwarm
BASIC-80 Rev. 5.21
[CP/M Version]
Copyright 1977, 78, 79, 80 (C) by Microsoft
Created: 15-Dec-80
30203 Bytes free

Message enabled on CP/M warm boot

CP/M WARM BOOT...

A>
```

The message will be active until the next reboot.

CP/M 2.2 AUTOEXEC

To enable the AUTOEXEC execution after the cold boot change the corresponding state to ON from the usual IOS boot selection menu.

To edit **AUTOEXEC.SUB** from drive A, you can use the **ED** editor. You can test the execution giving the command **SUBMIT AUTOEXEC** from drive A (you can omit the extension **.SUB** inside the **SUBMIT** command).

QP/M 2.71

To run QP/M 2.71 select it from the boot menu setting the Disk Set 1:

```

GtkTerm - /dev/ttyUSB0 9600-8-N-1
File Edit Log Configuration Control signals View
Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R100918

IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (17/09/18 19:49:13)
IOS: RTC DS3231 temperature sensor: 28C
IOS: CP/M Autoexec is OFF
IOS: Loading boot program (QPMLDR.BIN)... Done
IOS: Z80 is running from now

Z80-MBC2 QP/M 2.71 Cold Loader - S160918
Loading... done

Z80-MBC2 QP/M 2.71 BIOS - S150918
QP/M 2.71 Copyright 1985 (c) by MICROCode Consulting

A>d b: $t

File      Typ  Size  Created          Updated
-----
File      Typ  Size  Date   Time   Date   Time
-----
ART       TXT  16k  10-Sep-18  00:00  10-Sep-18  00:00
D         COM   4r  10-Sep-18  00:00  10-Sep-18  00:00
README   TXT   4k  10-Sep-18  00:00  10-Sep-18  00:00
SA       PAS   4k  10-Sep-18  00:00  10-Sep-18  00:00
TINST    COM  28k  10-Sep-18  00:00  10-Sep-18  00:00
TINST    DTA   8k  10-Sep-18  00:00  10-Sep-18  00:00
TINST    MSG   4k  10-Sep-18  00:00  10-Sep-18  00:00
TURBO    COM  32k  10-Sep-18  00:00  10-Sep-18  00:00
TURBO    MSG   4k  10-Sep-18  00:00  10-Sep-18  00:00
TURBO    OVR   4k  10-Sep-18  00:00  10-Sep-18  00:00
TURBOMSGOVR 4k  10-Sep-18  00:00  10-Sep-18  00:00

[ User 0 ] 11 Files Using 112 Kbytes of a 8176 Kbyte Drive
500 Directory Entries, with 8056 Kbytes Remaining on drive B:
A>

```

QP/M is an interesting alternative to CP/M developed by **MICROCode Consulting** that supports also file timestamping, and it is 100% CP/M 2.2 "compatible". MICROCode Consulting has released the original installation files and all the documentation in **their site** with the "restricted usage" condition, that means **free for non-commercial use** and for personal use only.

To enable timestamping (see upper screenshot) you need to install the optional RTC module.

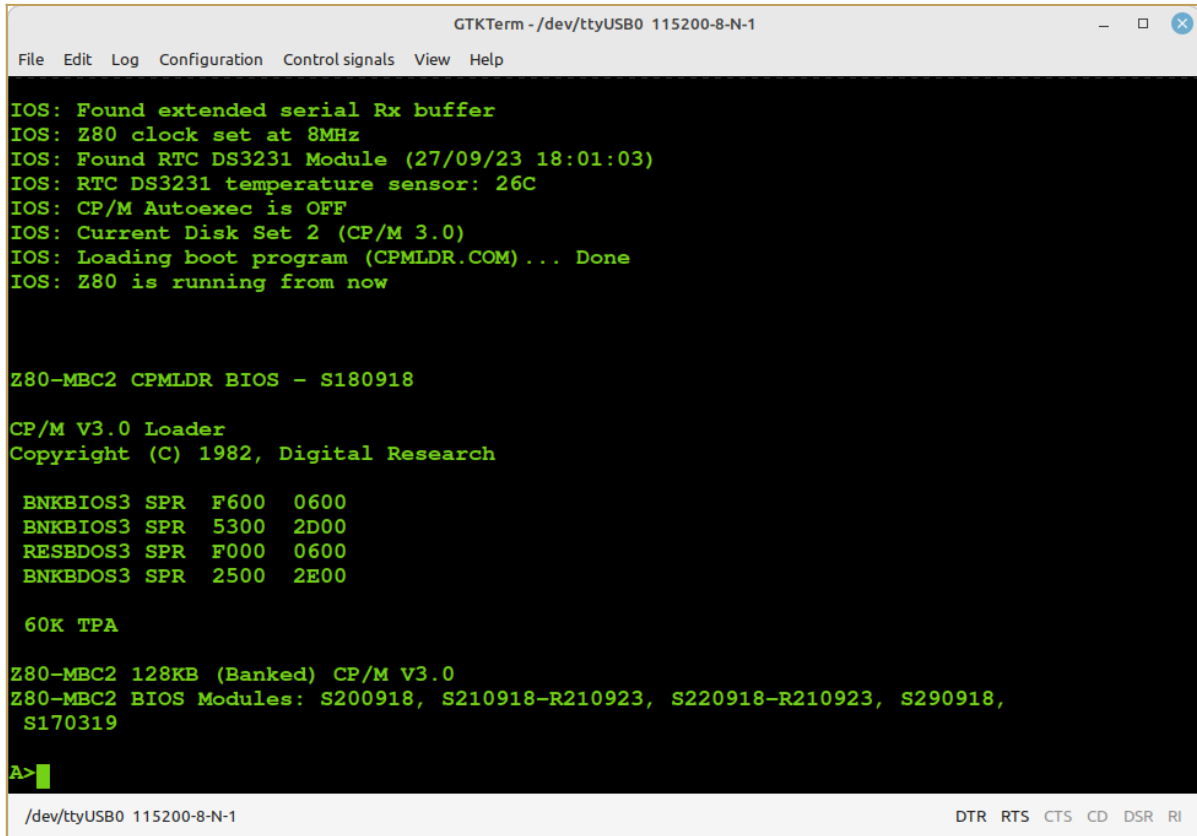
I suggest to read the QP/M documentation for the various commands (see the **Downloads** section in **their site**).

QP/M 2.71 AUTOEXEC

The QP/M uses for the batch file the **.QSB** extension. So the **AUTOEXEC** file is here named **AUTOEXEC.QSB**. To enable the AUTOEXEC execution after the cold boot change the corresponding state to ON from the usual IOS boot selection menu. In the drive A: there is an example of AUTOEXEC.QSB file ready to run.

CP/M 3

To run CP/M 3 select it from the boot menu setting the Disk Set 2:



```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (27/09/23 18:01:03)
IOS: RTC DS3231 temperature sensor: 26C
IOS: CP/M Autoexec is OFF
IOS: Current Disk Set 2 (CP/M 3.0)
IOS: Loading boot program (CPMLDR.COM) ... Done
IOS: Z80 is running from now

Z80-MBC2 CPMLDR BIOS - S180918

CP/M V3.0 Loader
Copyright (C) 1982, Digital Research

BNKBIOS3 SPR F600 0600
BNKBIOS3 SPR 5300 2D00
RESBDOS3 SPR F000 0600
BNKBDOS3 SPR 2500 2E00

60K TPA

Z80-MBC2 128KB (Banked) CP/M V3.0
Z80-MBC2 BIOS Modules: S200918, S210918-R210923, S220918-R210923, S290918,
S170319

A>
```

With CP/M 3.0 it is possible use the 128KB banked RAM to have a wider user area (TPA) for programs and a more "evoluted" OS.

Just as example of how it is easy with CP/M 3.0 manage multiple configurations, I've done also a "non-banked" 64KB version. The switch from one version to the other can be done simply running a batch from the console itself.

I've prepared two simple batch files to do that. From drive A: the command:

```
submit sys64
```

will set the 64KB "non-banked" version and then reboot the system.

To activate again the 128KB "banked" version give the command (from drive A.):

```
submit sys128
```


CP/M 3 AUTOEXEC

The AUTOEXEC switch for CP/M 3.0 works in a different way from the CP/M 2.2 and QP/M 2.71 implementations.

Now there is a custom utility (**AUTOEXEC**) that checks the IOS flag and sets the exit code accordingly (using the BDOS function 108). This allow to use the CP/M 3.0 batch conditional execution (see the ***CP/M 3 Programmer Guide*** par. 1.6.3) to run any wanted command or program based on the status of the IOS AUTOEXEC flag.

I've prepared an example using an other CP/M 3.0 feature, the "**PROFILE.SUB**" batch that is automatically executed at cold boot (if it exists). To activate it (in the drive A:) rename the file **PROFILE.SU** as **PROFILE.SUB** with the command:

```
ren profile.sub=profile.su
```

Now you can see how it works setting the AUTOEXEC flag on or off with the IOS "**Select boot mode or system parameters**" menu.

UCSD PASCAL

Thanks to ***Michel Bernard*** (a member of the **Z80-MBC2 User Group** on FB) who did the porting, now **UCSD Pascal** is running on the Z80-MBC2!

To run UCSD Pascal select it from the boot menu setting the Disk Set 3:

```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help

Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R280819

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found GPE Option
IOS: CP/M Autoexec is OFF

IOS: Select boot mode or system parameters:

0: No change (3)
1: Basic
2: Forth
3: Load OS from Disk Set 3 (UCSD Pascal)
4: Autoboot
5: iLoad
6: Change Z80 clock speed (->4MHz)
7: Toggle CP/M Autoexec (->ON)
8: Change Disk Set 3 (UCSD Pascal)

Enter your choice >8 Ok

Press CR to accept, ESC to exit or any other key to change
->Disk Set 3 (UCSD Pascal)
```

/dev/ttyUSB0 115200-8-N-1

DTR RTS CTS CD DSR RI

```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help

Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R280819

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found GPE Option
IOS: CP/M Autoexec is OFF
IOS: Current Disk Set 3 (UCSD Pascal)
IOS: Loading boot program (UCSDLDR.BIN)... Done
IOS: Z80 is running from now

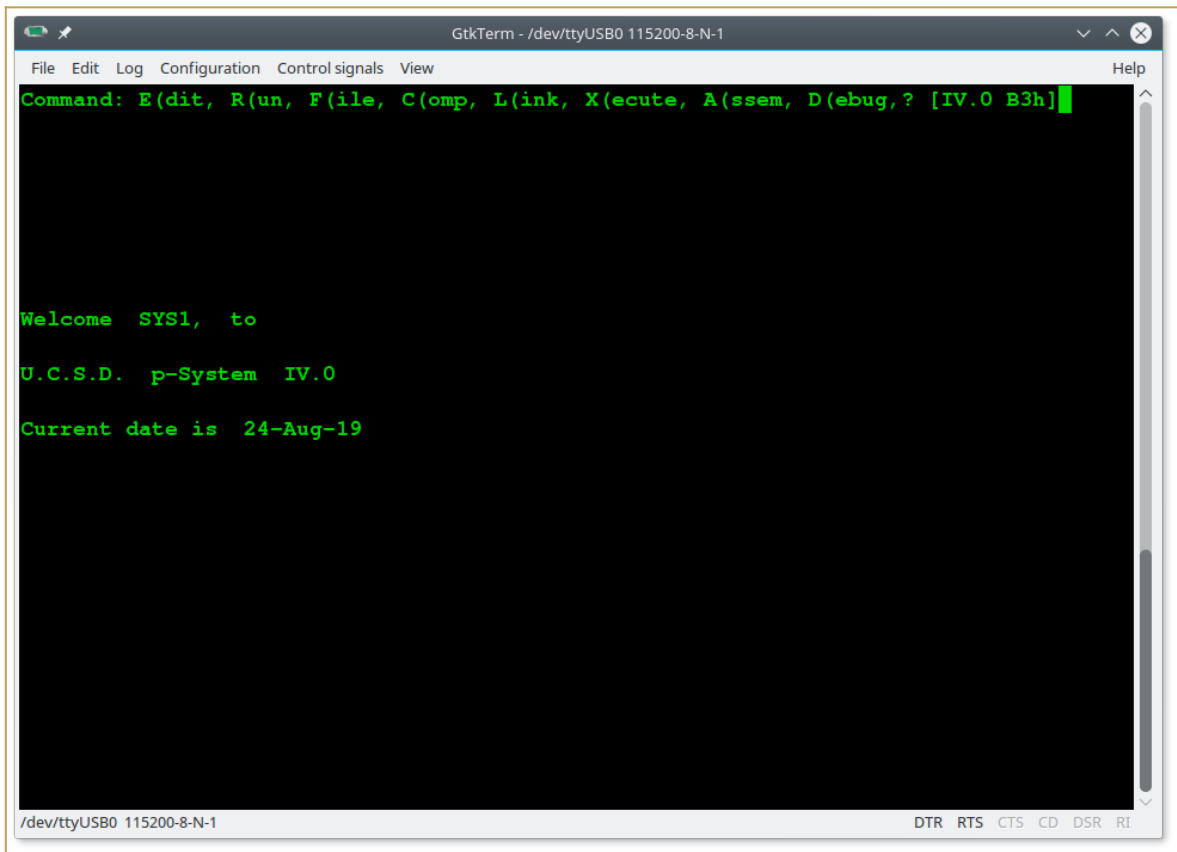
64K UCSD p-System IV.0 CBIOS V1.1 for Z80-MBC2, Copyright (C) 2019 by GmEsoft
Build: Aug 24 2019 - 21:57:34

Reading Secondary Bootstrap

Booting to UCSD Pascal
```

/dev/ttyUSB0 115200-8-N-1

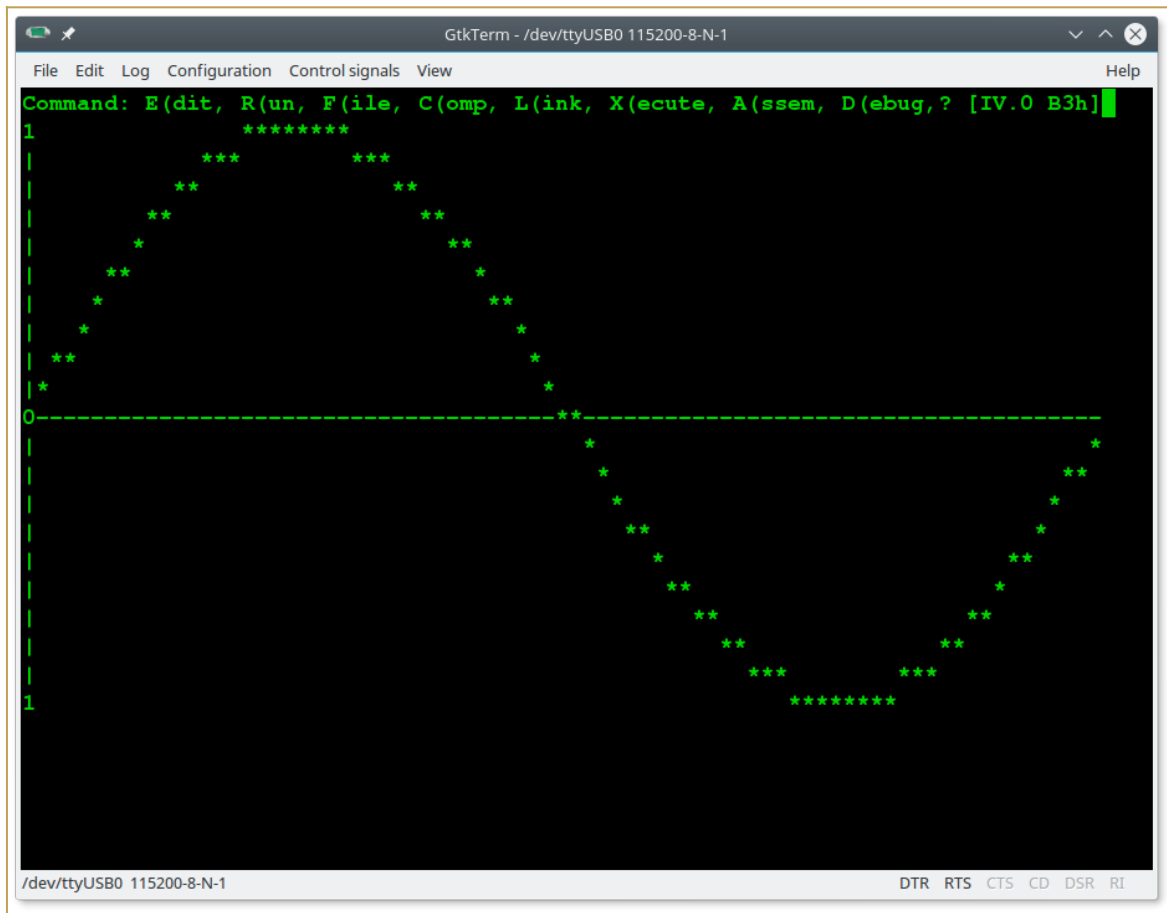
DTR RTS CTS CD DSR RI



In the SD image there are two volumes (disks) **SYS1:** and **SYS2:**


```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, ? [C.10]
SYS2:
LIBRARY.CODE      13 11-Aug- 8
DISKCHANGE.CODE   8 12-Aug- 8
DISKSIZE.CODE     3 12-Aug- 8
YALOE.CODE       12 12-Aug- 8
8080.ERRORS      8 12-Aug- 8
8080.OPCODES     3 12-Aug- 8
ASM8080.CODE     47 12-Aug- 8
DEBUGGER.CODE    21 12-Aug- 8
COPYDUPDIR.CODE  3 18-Aug- 8
MARKDUPDIR.CODE  4 18-Aug- 8
DECODE.CODE      28 18-Aug- 8
FINDPARAMS.CODE  9 18-Aug- 8
PATCH.CODE      33 18-Aug- 8
SCREENTEST.CODE  13 18-Aug- 8
COMPRESS.CODE    10 18-Aug- 8
CPMBOOT.CODE     22 18-Aug- 8
KERNEL.CODE     63 18-Aug- 8
RECOVER.G.CODE   8 18-Aug- 8
XREF.CODE        29 18-Aug- 8
CALC.TEXT        22 22-Aug- 8
SINE.TEXT        6 22-Aug- 8
CALC.CODE        7 22-Aug- 8
SINE.CODE        2 22-Aug- 8
23/23 files<listed/in-dir>, 380 blocks used, 2052 unused, 2052 in largest
/dev/ttyUSB0 115200-8-N-1 DTR RTS CTS CD DSR RI
```

Here the execution of an example (SINE.CODE) already compiled on the SYS2: disk:

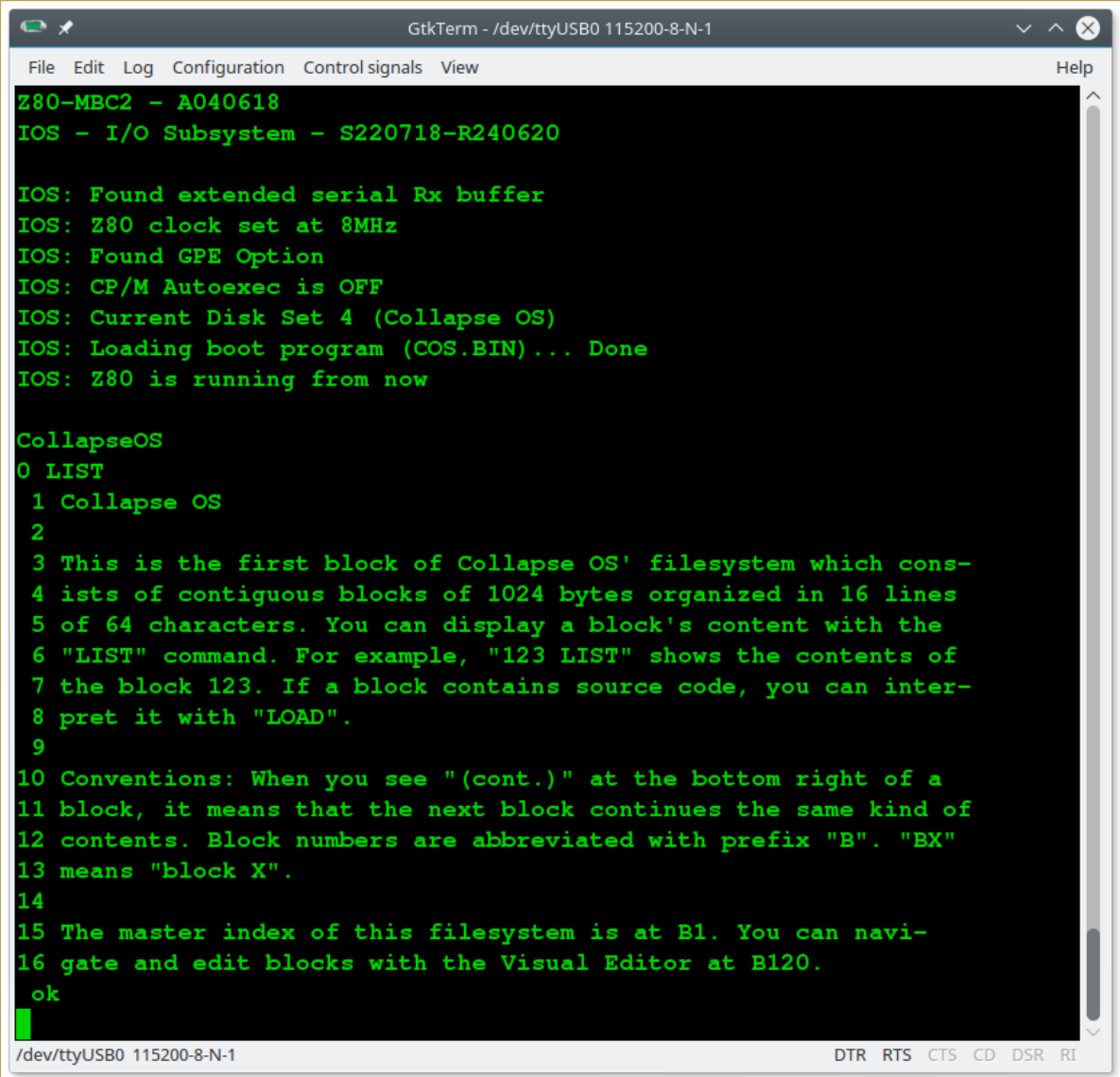


In the folder "**UCSD Pascal**" inside the SD there are the original files and sources provided by Michel Bernard (https://github.com/GmEsoft/Z80-MBC2_UCSDP).

A lot of documentation and books about UCSD Pascal can be found [here](#).

COLLAPSE OS

To run Collapse OS select it from the boot menu setting the Disk Set 4:



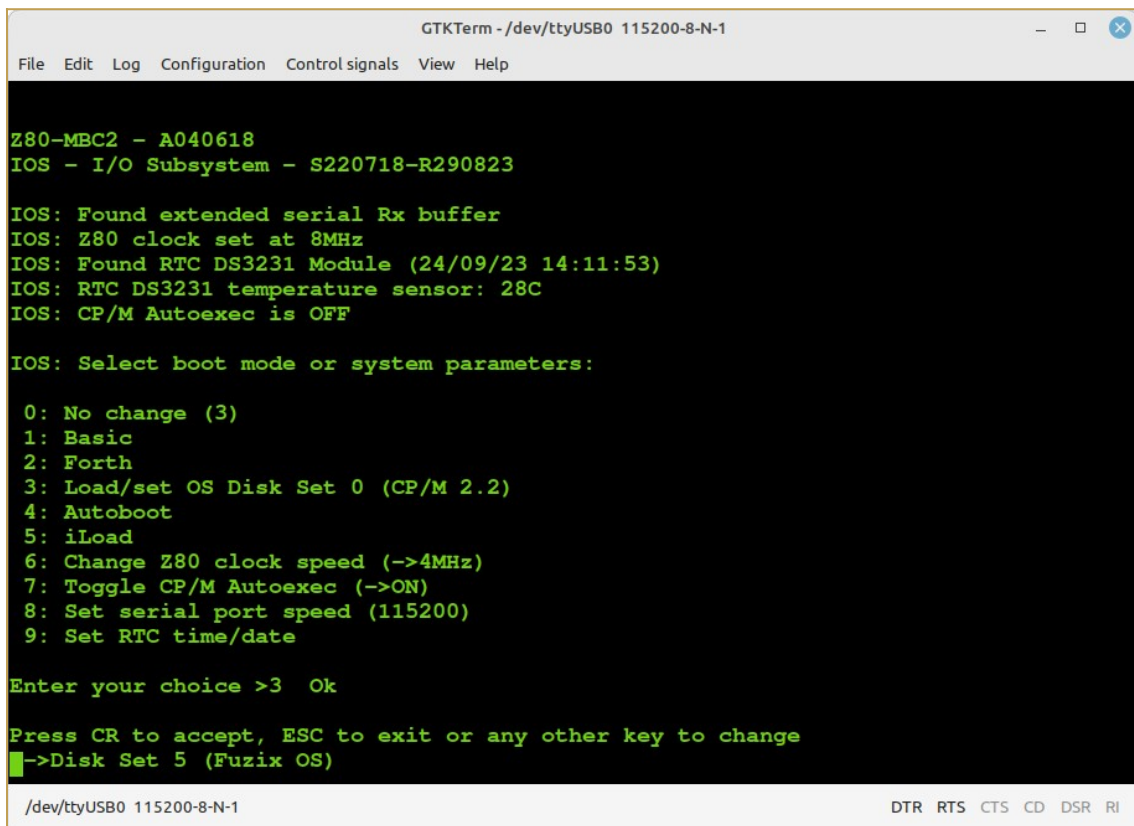
```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R240620
IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found GPE Option
IOS: CP/M Autoexec is OFF
IOS: Current Disk Set 4 (Collapse OS)
IOS: Loading boot program (COS.BIN)... Done
IOS: Z80 is running from now

CollapseOS
0 LIST
 1 Collapse OS
 2
 3 This is the first block of Collapse OS' filesystem which cons-
 4 ists of contiguous blocks of 1024 bytes organized in 16 lines
 5 of 64 characters. You can display a block's content with the
 6 "LIST" command. For example, "123 LIST" shows the contents of
 7 the block 123. If a block contains source code, you can inter-
 8 pret it with "LOAD".
 9
10 Conventions: When you see "(cont.)" at the bottom right of a
11 block, it means that the next block continues the same kind of
12 contents. Block numbers are abbreviated with prefix "B". "BX"
13 means "block X".
14
15 The master index of this filesystem is at B1. You can navi-
16 gate and edit blocks with the Visual Editor at B120.
ok
/dev/ttyUSB0 115200-8-N-1 DTR RTS CTS CD DSR RI
```

For more info the *Collapse OS site* is [here](#).

FUZX OS

To run Fuzix OS select it from the boot menu setting the Disk Set 5:



```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help

Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R290823

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (24/09/23 14:11:53)
IOS: RTC DS3231 temperature sensor: 28C
IOS: CP/M Autoexec is OFF

IOS: Select boot mode or system parameters:

0: No change (3)
1: Basic
2: Forth
3: Load/set OS Disk Set 0 (CP/M 2.2)
4: Autoboot
5: iLoad
6: Change Z80 clock speed (->4MHz)
7: Toggle CP/M Autoexec (->ON)
8: Set serial port speed (115200)
9: Set RTC time/date

Enter your choice >3 Ok

Press CR to accept, ESC to exit or any other key to change
->Disk Set 5 (Fuzix OS)

/dev/ttyUSB0 115200-8-N-1 DTR RTS CTS CD DSR RI
```

You have to select **hda2** as **bootdev** device when asked, and then log as **root** user (no password):


```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
IOS - I/O Subsystem - S220718-R290823

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 8MHz
IOS: Found RTC DS3231 Module (24/09/23 13:04:08)
IOS: RTC DS3231 temperature sensor: 28C
IOS: CP/M Autoexec is OFF
IOS: Current Disk Set 5 (Fuzix OS)
IOS: Loading boot program (FUZIX.BIN)... Done
IOS: Z80 is running from now

FUZIX version 0.4
Copyright (c) 1988-2002 by H.F.Bower, D.Braun, S.Nitschke, H.Peraza
Copyright (c) 1997-2001 by Arcady Schekochikhin, Adriano C. R. da Cunha
Copyright (c) 2013-2015 Will Sowerbutts <will@sowerbutts.com>
Copyright (c) 2014-2023 Alan Cox <alan@etchedpixels.co.uk>
Devboot
128kB total RAM, 64kB available to processes (15 processes max)
Enabling interrupts ... ok.
hda: hda1 (swap) hda2
bootdev: hda2
Mounting root fs (root_dev=2, ro): OK
Starting /init
27 buffers added
init version 0.9.1
Checking root file system.

^ ^
n n   Fuzix 0.4
>@<

      Welcome to Fuzix

m m

login: root

Welcome to FUZIX.
# banner Z80-MBC2
ZZZZZZZ 88888 000      M      M BBBBBB  CCCC  22222
  Z 8      8 0 0      MM      MM B      B C      C 2      2
  Z 8      8 0 0 0    M M M M B      B C      C      2
  Z 88888 0 0 0 ---- M M  M BBBBBB  C      22222
  Z 8      8 0 0 0    M      M B      B C      2
  Z 8      8 0 0 0    M      M B      B C      C 2
ZZZZZZZ 88888 000      M      M BBBBBB  CCCC  222222

# █

/dev/ttyUSB0 115200-8-N-1                                DTR RTS CTS CD DSR RI
```

Remember to give the **shutdown** command before powering off to avoid the file system warning and checking at the next Fuzix reboot:

```

GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
IOS: Z80 is running from now

FUZIX version 0.4
Copyright (c) 1988-2002 by H.F.Bower, D.Braun, S.Nitschke, H.Peraza
Copyright (c) 1997-2001 by Arcady Schekochikhin, Adriano C. R. da Cunha
Copyright (c) 2013-2015 Will Sowerbutts <will@sowerbutts.com>
Copyright (c) 2014-2023 Alan Cox <alan@etchedpixels.co.uk>
Devboot
128kB total RAM, 64kB available to processes (15 processes max)
Enabling interrupts ... ok.
hda: hda1 (swap) hda2
bootdev: hda2
Mounting root fs (root_dev=2, ro): warning: mounting dirty file system, forcing r/o.
OK
Starting /init
27 buffers added
init version 0.9.1
Checking root file system.
Filesystem was not cleanly unmounted.
Device 0 has fsize = 12288 and isize = 256. Continue? n

^ ^
n n Fuzix 0.4
>@<
Welcome to Fuzix
m m

login: █

/dev/ttyUSB0 115200-8-N-1 DTR RTS CTS CD DSR RI

```

```

GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help
-rwxr-xr-x 1 root root 13447 Aug 18 14:12 tail
-rwxr-xr-x 1 root root 14732 Aug 18 14:12 tar
-rwxr-xr-x 1 root root 1101 Aug 18 14:12 tee
-rwxr-xr-x 1 root root 2138 Aug 18 14:12 telinit
-rwxr-xr-x 1 root root 11992 Aug 18 14:12 termcap
-rwxr-xr-x 1 root root 10817 Aug 18 14:12 tget
-rwxr-xr-x 1 root root 6516 Aug 18 14:12 touch
-rwxr-xr-x 1 root root 1674 Aug 18 14:12 tr
-rwxr-xr-x 1 root root 294 Aug 18 14:12 true
-rwxr-xr-x 1 root root 8250 Aug 18 14:12 ue
-rwxr-xr-x 1 root root 12234 Aug 18 14:12 umount
-rwxr-xr-x 1 root root 3507 Aug 18 14:12 uname
-rwxr-xr-x 1 root root 9911 Aug 18 14:12 uniq
-rwxr-xr-x 1 root root 16342 Aug 18 14:12 uptime
-rwxr-xr-x 1 root root 14859 Aug 18 14:12 uud
-rwxr-xr-x 1 root root 8854 Aug 18 14:12 uue
-rwxr-xr-x 2 root root 13833 Aug 18 14:12 vi
-rwxr-xr-x 2 root root 13833 Aug 18 14:12 vile
-rwxr-xr-x 1 root root 8570 Aug 18 14:12 wc
-rwxr-xr-x 1 root root 5429 Aug 18 14:12 which
-rwxr-xr-x 1 root root 14100 Aug 18 14:12 who
-rwxr-xr-x 1 root root 2633 Aug 18 14:12 whoami
-rwxr-xr-x 1 root root 9365 Aug 18 14:12 write
-rwxr-xr-x 1 root root 9607 Aug 18 14:12 xargs
-rwxr-xr-x 1 root root 402 Aug 18 14:12 yes
# shutdown
Halted.
█

/dev/ttyUSB0 115200-8-N-1 DTR RTS CTS CD DSR RI

```

For more info the *Fuzix OS site* is *here*.

HOW BUILD FUZIX OS FROM SCRATCH

Building Fuzix OS (<https://github.com/EtchedPixels/FUZIX>) for a Z80 CPU with banked RAM **requires** a special patched version of SDDC available from here: <https://github.com/EtchedPixels/sdcc280> .

This special SDDC version is source only, and it needs to be compiled.

These instructions have been written using an Ubuntu based Linux distro (Linux Mint 20.3). Aside from the package installation commands, the rest of the steps should work with many other Linux distributions as well.

STEP 1:

First, install the necessary packages, if they are not already there (they should...):

```
sudo apt-get install gcc
sudo apt-get install build-essential
sudo apt-get install automake gputils flex texinfo bison byacc
```

STEP 2:

Install the following package that is not usually present on a Ubuntu based distro:

```
sudo apt-get install libboost-all-dev
```

STEP 3:

Now get the special version of the SDDC compiler for Fuzix:

```
git clone https://github.com/EtchedPixels/sdcc280.git
```

and give the following commands to compile and install it:

```
cd sdcc280
cd sdcc
./configure
make
sudo make install
cd ../../
```

STEP 4:

Get Fuzix source code:

```
git clone https://github.com/EtchedPixels/FUZIX.git
cd FUZIX
```

STEP 5:

Modify the Makefile and change the line that says TARGET=..... to be TARGET=z80-mbc2 and compile it (it can take hours...):

```
sudo make
```

STEP 6:

Now create the fuzix.bin file and the virtual disk file (.DSK) with the command (from the same directory):

```
sudo make diskimage
```

All done! In the folder FUZIX/Images/z80-mbc2 you'll get:

fuzix.bin

DS0N01.DSK

Rename DS0N01.DSK as DS5N01.DSK and copy both files into the SD of the Z80-MBC2 SBC.

NOTE: This guide was adapted from here: <http://www.forofpga.es/viewtopic.php?t=422>

* * USING THE SDCC CROSS COMPILER

* *

Using the SDCC (Small Device C Compiler) cross-compiler it is possible setup a toolchain to program the Z80-MBC2 with the C language, doing all the development on a PC and uploading the code with the serial port and then executing it on the target Z80-MBC2 with *iLoad*.

SDCC can be found here: <https://sdcc.sourceforge.net/>.

After installing it, SDCC needs to be in some way instructed about how to deal with the specific HW of the Z80-MBC2.

For this reason in the SD image, inside the `\SDCC` folder, there are two support files: **`S190818-R011023_crt0.s`** and **`S290923_Z80-MBC2.c`**.

All the steps needed to configure the toolchain are explained below (***we will assume a Windows operating system here***, but the steps are similar for Linux):

STEP 1:

Copy the two support files **`S190818-R011023_crt0.s`** and **`S290923_Z80-MBC2.c`** from the SD image (`\SDCC` folder) to your *working directory* (it is the folder where your C source files are stored) and compile the first file with the command (from your working directory):

```
sdasz80 -plogfff -o S190818-R011023_crt0.s
```

It will be created the **`S190818-R011023_crt0.rel`** file.

STEP 2:

Now it's time to compile the second support file (**`S290923_Z80-MBC2.c`**). Here things are a little more complex because this file can be compiled in two different ways which differ depending on whether ***interrupts are enabled or not***.

The need to have interrupts enabled or not depends on whether your user program uses them or not.

To enable the interrupts support compile with the command (from your working directory):

```
sdcc -c -mz80 -DZ80MBC2IRQ S290923_Z80-MBC2.c
```

Instead to disable the interrupts support compile with (from your working directory):

```
sdcc -c -mz80 S290923_Z80-MBC2.c
```

It will be created the **S290923_Z80-MBC2.rel** file.

STEP 3:

iLoad uses the first address as starting address for the execution, so the executable file (Intel-Hex formatted) must be in ascending address order. This is not guaranteed by SDCC, so you need to use the **srec_cat** utility to order the file. You can download this utility from here: <https://srecord.sourceforge.net/> and then you have to copy the **srec_cat.exe** file into your working directory.

All done!

To compile your source file the command is (from your working directory):

```
sdcc -mz80 --no-std-crt0 S190818-R011023_crt0.rel <your_source.c> S290923_Z80-MBC2.rel -o out.hex
```

It will be created the **out.hex** file (Intel-hex formatted executable file).

Now to order the file give the command (from your working directory):

```
srec_cat -disable-sequence-warnings out.hex -Intel -o load.hex -Intel
```

This will create the ordered file ready to be loaded with iLoad: **load.hex**.

Now you can upload and execute **load.hex** using the iLoad boot mode of the Z80-MBC2.

SDCC: SETTING UP AN AUTOMATED TOOLCHAIN (WINDOWS)

To create an automated toolchain you need another "ingredient", a terminal emulator supporting scripts. Here we will use **Tera Term**. You can download Tera Term from here: <https://ttssh2.osdn.jp/index.html.en>.

After installing Tera Term, from the SD image inside the **SDCC** folder, copy into the working directory the following batch files: **SDC.BAT** and **L.BAT**.

Before using the **L.BAT** batch file you have to adapt two parameters according with the configuration of your PC.

Go at line 18 and verify the path where Tera Term (ttermpro.exe) is installed, and at line 19 the number of the COM port used to connect the Z80-MBC2 to your PC.

You need also to copy the Tera Term script **LoadZ80.ttl** from the **/SDCC** folder (inside the SD image) to the directory where Tera Term (ttermpro.exe) is installed, and adapt the parameter at line 15 with the complete path of your working directory in your system.

Now to compile ***your_source.c*** file give the command (from your working directory):

```
SDC your_source.c
```

and to upload and execute it on the Z80-MBC2 (from your working directory):

```
L
```

Remember to close the Tera Term window before executing the L.BAT command again.

SDCC: SETTING UP AN AUTOMATED TOOLCHAIN (LINUX)

On Linux the procedure from *STEP 1* to *STEP 3* is nearly the same. It is possible to install easily the ***srec_cat*** utility.

About the terminal emulator, on Linux you can use ***minicom***.

The needed script file must be created. So create a text file named ***minicom.mac*** with the following text:

```
#  
# Minicom script for the automated SDCC toolchain for the Z80-MBC2  
#  
sleep 3  
! ascii-xfr -s -n -l 100 load.hex  
sleep 1
```

The command to upload the executable ***load.hex*** file (Intel-Hex formatted) into the target Z80-MBC2 and execute it using ***minicom*** is:

```
minicom -w -S minicom.mac -D /dev/ttyUSB0
```

where ***/dev/ttyUSB0*** must be adapted to the port you are using to connect the Z80-MBC2 on your system.

Remember to close ***minicom*** before the next upload with the ***Alt-A*** key followed by ***X***.

SDCC: USING AUTOBOOT

If you want make your custom .hex executable "permanent", you can use the ***Autoboot*** mode of the Z80-MBC2.

First you need another utility, **hex2bin.exe**, in your working directory. You can find hex2bin **here**.

Then with the command:

```
hex2bin -p 00 out.hex
```

your **out.hex** executable file (Intel-Hex formatted) will be converted in a flat binary file **out.bin**.

At this point rename **out.bin** as **autoboot.bin** and copy it into the root of the SD used by the Z80-MBC2.

Now selecting the **Autoboot** mode from the Z80-MBC2 boot menu will automatically run it when you turn on the board (or after a reset).

SDCC: EXAMPLES

In the SD image (**\SDCC\examples** folder) there are a few sources examples to test the toolchain.

Remember that you need to re-compile the **S290923_Z80-MBC2.c** file as explained in the STEP 2 every time you switch from a program requiring interrupts enabled to another one wanting them disabled and vice versa.

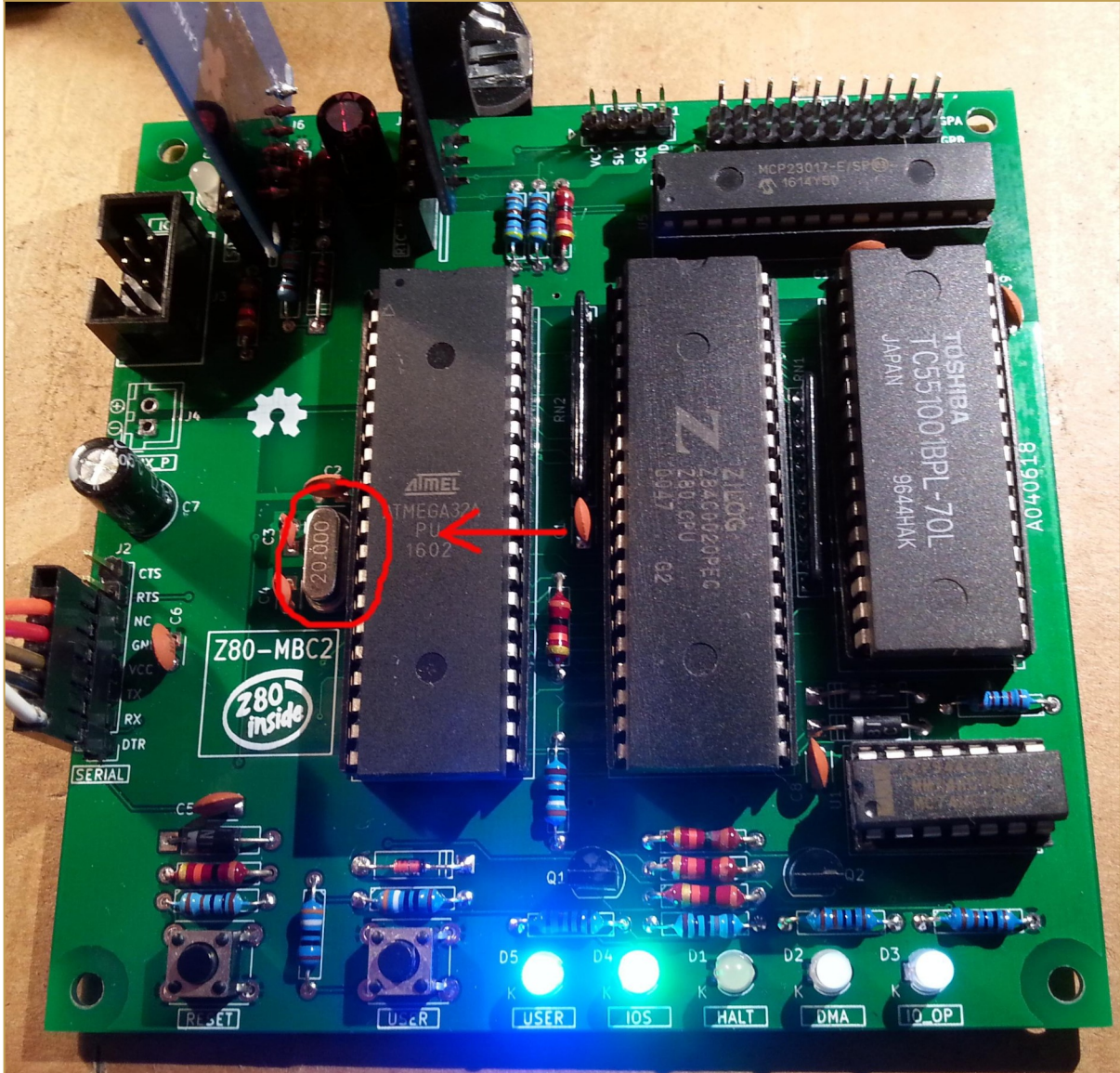
I suggest to take a look at the content of the **Blink_MBC2.c** and **Blink_MBC2_IRQ.c** examples, and at the source of both the support files (**S190818-R011023 crt0.s** and **S290923_Z80-MBC2.c**) to understand how they works.

Dealing with the *IOS Opcodes* requires that you read the various comments on the IOS source file (.ino) explaining how they works.

Using IOS Opcodes when interrupts are enabled requires that you treat them as an atomic operation, disabling interrupts before the Opcode call and re-enabling them immediately afterwards (see the content of the previous source files as an example).

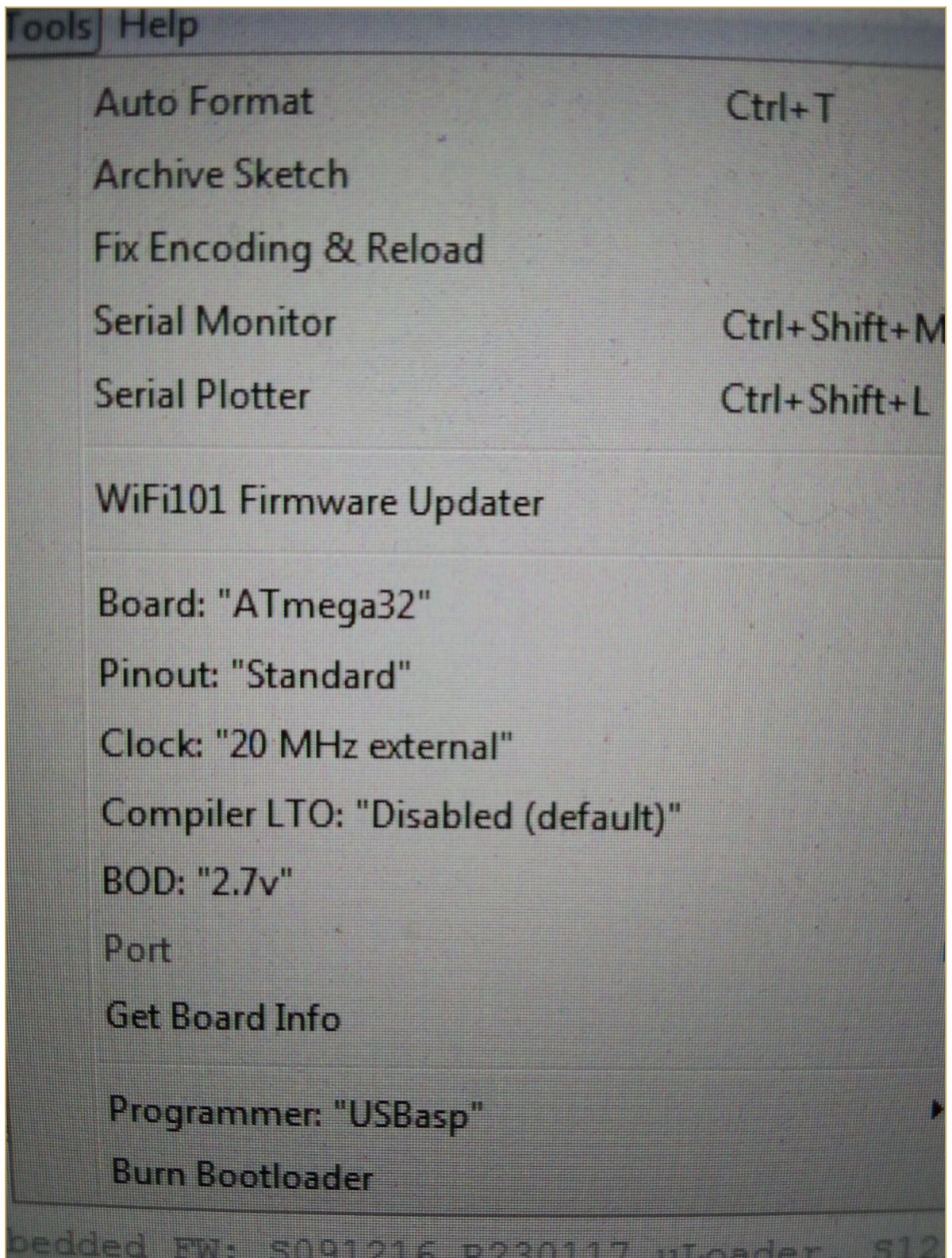
* * OVERCLOCKING THE Z80-MBC2 * *

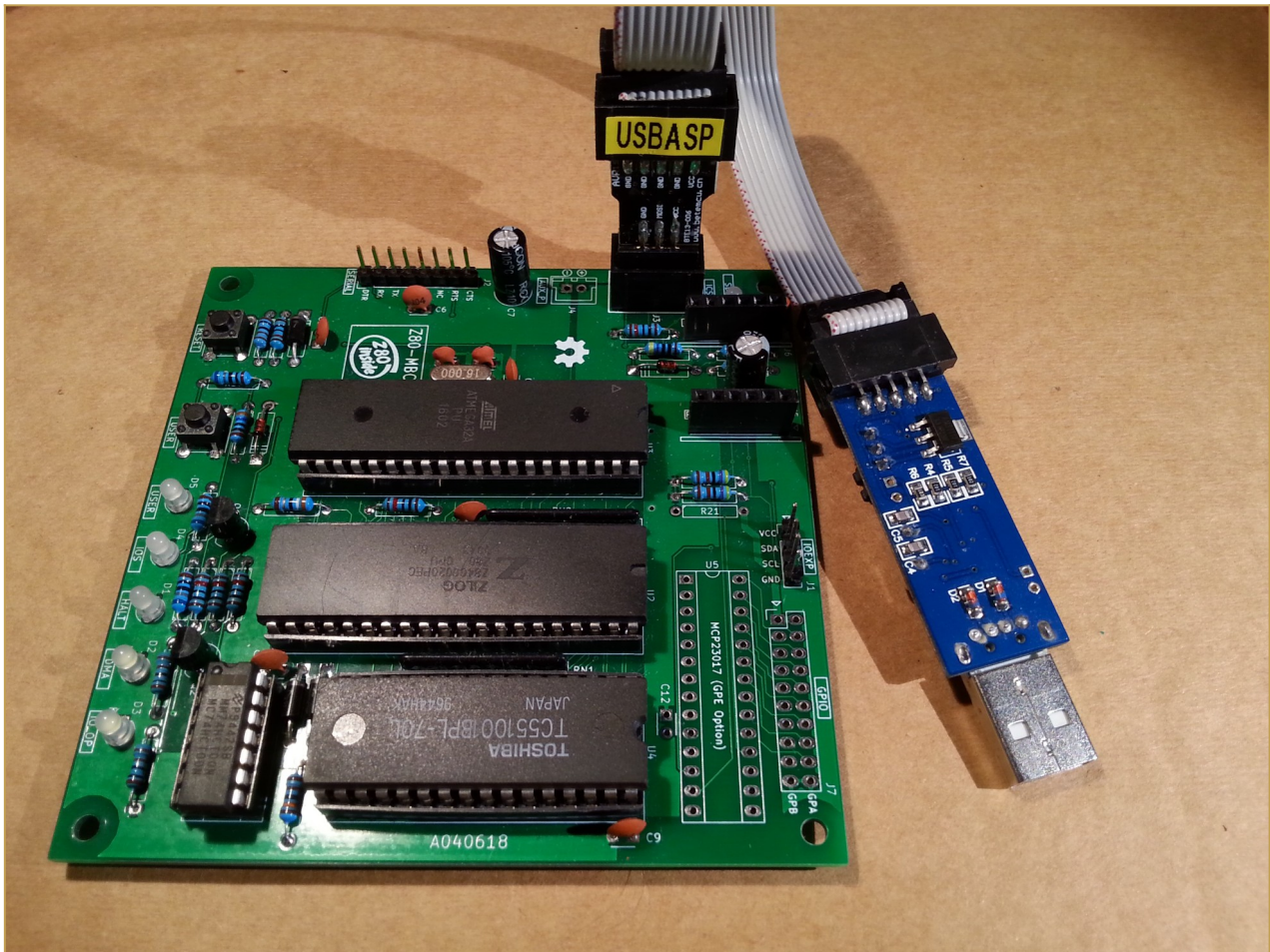
Since the **Mighty Core** gives the chance to choice a 20MHz bootloader, I've decided to try to "overclock" the Atmega32A using a 20MHz quartz:



You don't need others HW changes, just use a 20MHz quartz instead of a 16MHz one. The Z80 clock speed will be at 10MHz.

You have to select the "20MHz external" option in the "Toos" menu of Arduino IDE before flashing the 20MHz bootloader:





Of course you need to load the sketch again (using the "20MHz external" option). IOS will display the new clock speed:

```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Control signals View Help

Z80-MBC2 - A040618
IOS - I/O Subsystem - S220718-R290823

IOS: Found extended serial Rx buffer
IOS: Z80 clock set at 10MHz
IOS: Found GPE Option
IOS: CP/M Autoexec is OFF
IOS: Current Disk Set 0 (CP/M 2.2)
IOS: Loading boot program (CPM22.BIN)... Done
IOS: Z80 is running from now

Z80-MBC2 CP/M 2.2 BIOS - S030818-R120923
CP/M 2.2 Copyright 1979 (c) by Digital Research

A>d
ASCIART  BAS  4k : ASM      COM  8k : AUTOEXEC SUB  4k : AUTOEXEC TXT  4k
CATCHUM  COM 32k : CATCHUM DAT  4k : CATCONF  COM 24k : D          COM  4k
DDT      COM  8k : DUMP    COM  4k : ED       COM  8k : GENHEX  COM  4k
GORILLA  COM 32k : GPELED  BAS  4k : GPIO    BAS  4k : HELLO   ASM  4k
HELLO    COM  4k : LADCONF COM 24k : LADDER  COM 40k : LADDER  DAT  4k
LOAD     COM  4k : MAC     COM 16k : MBASIC  COM 24k : MBASIC85 COM 24k
PEG      COM  8k : PIP     COM  8k : RTC     BAS  4k : SHOWWARM BAS  4k
SPP      BAS  4k : STARTREK BAS 24k : STAT    COM  8k : SUBMIT  COM  4k
TREKINST BAS  8k : USERLED BAS  4k : XMODEM  CFG  8k : XMODEM  COM  8k
XSUB     COM  4k : ZDE16   COM 20k : ZDENST16 COM 12k
A: Total of 420k in 39 files with 7740k space remaining.
A>
```

Remember that using a 20MHz quartz you are **out of the Atmega32a specifications** (the Atmega32a is rated at 16MHz max.), so you are in a "grey area" where things "may works"...

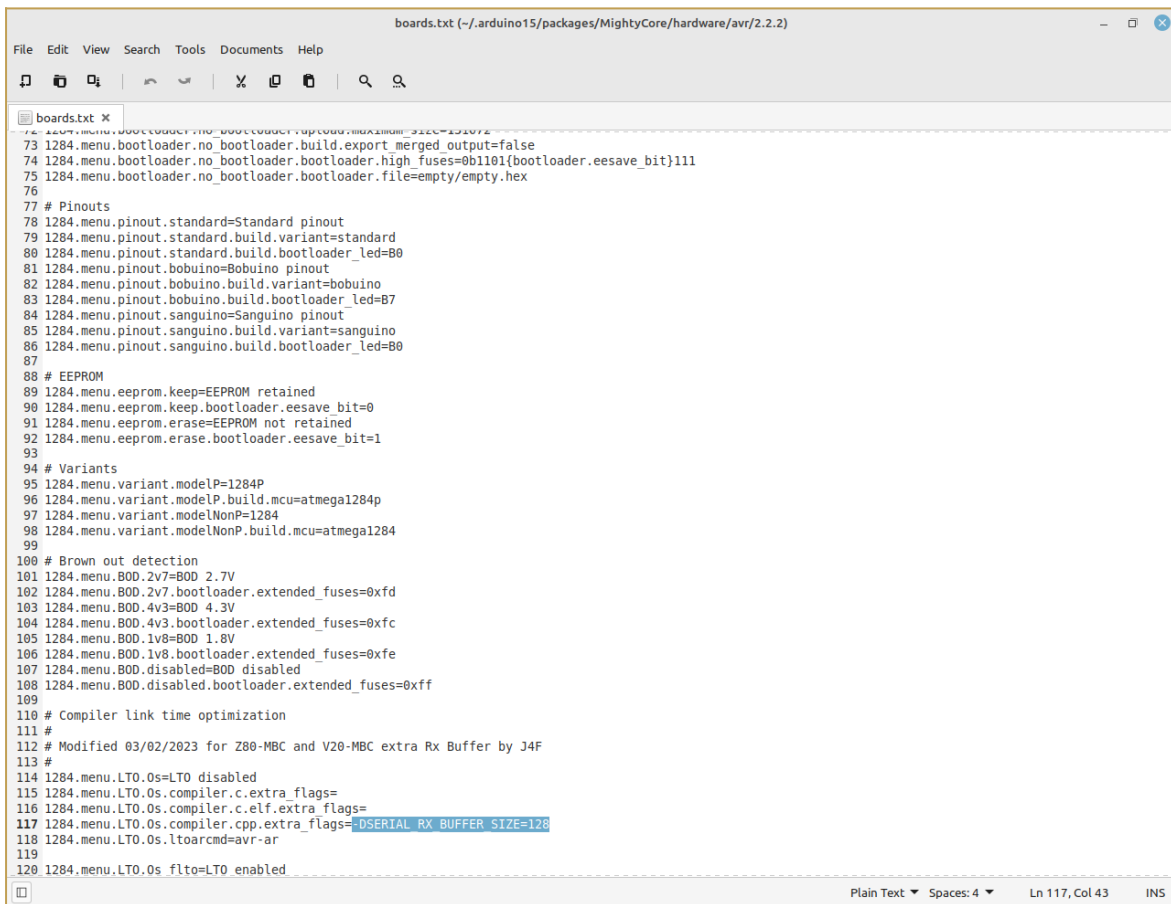
** USING AN

ATMEGA1284/ATMEGA1284P **

Starting with IOS S220718-R290823 it is possible to use an **Atmega1284/Atmega1284P** as MCU. This will let more space for any customization.

Of course you have to re-compile the IOS source selecting the right MCU.

Remember to apply the changes described in the paragraph "*HOW ENABLE THE EXTENDED RX BUFFER FOR XMODEM (CP/M)*" searching for the Atmega1284 section:



```
boards.txt (-/.arduino15/packages/MightyCore/hardware/avr/2.2.2)
File Edit View Search Tools Documents Help
boards.txt x
72 1284.menu.bootloader.no_bootloader.build.export_merged_output=false
73 1284.menu.bootloader.no_bootloader.build.export_merged_output=false
74 1284.menu.bootloader.no_bootloader.bootloader.high_fuses=0b1101{bootloader.eesave_bit}111
75 1284.menu.bootloader.no_bootloader.bootloader.file=empty/empty.hex
76
77 # Pinouts
78 1284.menu.pinout.standard=Standard pinout
79 1284.menu.pinout.standard.build.variant=standard
80 1284.menu.pinout.standard.build.bootloader_led=B0
81 1284.menu.pinout.bobuino=Bobuino pinout
82 1284.menu.pinout.bobuino.build.variant=bobuino
83 1284.menu.pinout.bobuino.build.bootloader_led=B7
84 1284.menu.pinout.sanguino=Sanguino pinout
85 1284.menu.pinout.sanguino.build.variant=sanguino
86 1284.menu.pinout.sanguino.build.bootloader_led=B0
87
88 # EEPROM
89 1284.menu.eeprom.keep=EEPROM retained
90 1284.menu.eeprom.keep.bootloader.eesave_bit=0
91 1284.menu.eeprom.erase=EEPROM not retained
92 1284.menu.eeprom.erase.bootloader.eesave_bit=1
93
94 # Variants
95 1284.menu.variant.modelP=1284P
96 1284.menu.variant.modelP.build.mcu=atmega1284p
97 1284.menu.variant.modelNonP=1284
98 1284.menu.variant.modelNonP.build.mcu=atmega1284
99
100 # Brown out detection
101 1284.menu.BOD.2v7=BOD 2.7V
102 1284.menu.BOD.2v7.bootloader.extended_fuses=0xfd
103 1284.menu.BOD.4v3=BOD 4.3V
104 1284.menu.BOD.4v3.bootloader.extended_fuses=0xfc
105 1284.menu.BOD.1v8=BOD 1.8V
106 1284.menu.BOD.1v8.bootloader.extended_fuses=0xfe
107 1284.menu.BOD.disabled=BOD disabled
108 1284.menu.BOD.disabled.bootloader.extended_fuses=0xff
109
110 # Compiler link time optimization
111 #
112 # Modified 03/02/2023 for Z80-MBC and V20-MBC extra Rx Buffer by J4F
113 #
114 1284.menu.LTO.0s=LTO disabled
115 1284.menu.LTO.0s.compiler.c.extra_flags=
116 1284.menu.LTO.0s.compiler.c.elf.extra_flags=
117 1284.menu.LTO.0s.compiler.cpp.extra_flags=-D SERIAL_RX_BUFFER_SIZE=128
118 1284.menu.LTO.0s.ltoarcmd=avr-ar
119
120 1284.menu.LTO.0s.flto=LTO enabled
Plain Text Spaces: 4 Ln 117, Col 43 INS
```

*** * PROJECT STATUS * ***

Currently both IOS-LITE and IOS are available. The first is a simplified version that doesn't support the SD, the second is a full featured version that requires the SD module (e. g. to run CP/M).

The current revision of IOS allows you to run **CP/M 2.2, CP/M 3.0, QP/M 2.71, UCSD Pascal, Collapse OS** and **Fuzix OS** (and the stand-alone versions of Basic and Forth, the same supported by IOS-LITE) with 16 virtual disks (8Mbytes each) for each OS.

Support files for the **SDCC cross-compiler** have been added inside the SD image (/SDCC folder), including interrupt handling.

The add-on board **uTerm** has been released.

The add-on board **uCom** has been released.

The **SPP Adapter board** (parallel printer standard interface) is supported under CP/M 2.2 and CP/M 3 (banked).

Not suited for aerospace applications! ☹

*** * HOW TO GET A PCB * ***

Because some people asked about this, I've prepared an "easy" link to get a small lot (5 pcs minimum) of PCB. The link is **this one**.

*** * HOW TO GET A KIT OR AN ASSEMBLED UNIT * ***

If you are looking for a kit with all the needed parts or an assembled unit ready to use now there is a professional seller that can sell both and ship worldwide.

The link to the seller is **this one**.

* * Z80-MBC2 USER GROUP * *

An "User Group" was created on Facebook:

<https://www.facebook.com/groups/Z80MBC2>.

* * LICENSING AND CREDITS * *

All the project files (SW & HW) are licensed under GPL v3.

If you use this material in any way a reference to the author (me ☺) will be appreciated.

CP/M seems to be Open Source now (see [here](#)).

PetitFS was developed by **ChaN**.

Basic stand-alone interpreter was an adaptation from **Grant Searle** work.

Forth stand-alone interpreter was originally ported to the **Z80-MBC** by **Bill Westfield**.

UCSD Pascal was ported by **Michel Bernard**.

Collapse OS was designed and ported by **Virgil Dupras**.

Fuzix OS was designed and ported by **Alan Cox** (www.fuzix.org).

* * NOTES ON THIS MANUAL * *

This manual has been extracted from the *Z80-MBC2 project page on Hackaday.io*, so it should be considered as a "frozen image".

The table of contents has hyperlinks so it's possible to jump easily to the desired chapter/paragraph, and all the original hyperlinks have been maintained inside the text.

All the images are hyperlinked to the original one on the web pages.

Of course to check latest changes it is always better to take a look on the real site.