dastaZ80 Mark II
Technical Reference Manual

Disclaimer

The products described in this manual are intended for educational purposes, and should not be used for controlling any machinery, critical component in life support devices or any system in which failure could result in personal injury if any of the described here products fail.

These products are subject to continuous development and improvement. All information of a technical nature and particulars of the products and its use are given by the author in good faith. However, it is acknowledged that there may be errors or omissions in this manual. Therefore, the author cannot accept any liability for any loss or damage arising from the use of any information or particulars in this manual.

Licenses

Document Conventions

The following conventions are used in this manual:

DEVICE	Device names are displayed in bold all upper case letters, and refer to hardware devices.
`Courier`	Text appearing in the `Courier` font represents either an OS System Variable a Z80 CPU Register or a Z80 Flag. OS System Variables are identifiers for specific **MEMORY** addresses that can be used to read statuses and to pass information between routines or programs.
`0x14B0`	Numbers prefixed by 0x indicate an Hexadecimal value. Unless specified, memory addresses are always expressed in Hexadecimal.

The SD card is referred as **DISK**.

The Floppy Disk Drive is referred as **DISK** or as **FDD**.

The 80 column text VGA output is referred as **CONSOLE** or as **High Resolution Display**.

The 40 column graphics Composite Video output is referred as **Low Resolution Display**.

The Operating System may be referred as DZOS, dzOS or simply OS.

MEMORY refers to both **ROM** and **RAM**.

Memory used by the **Low Resolution Display** is referred as **VRAM** (Video RAM).

The sound chip may be referred as **Sound Chip** or **PSG** (Programmable Sound Generator).

Related Documentation

- dastaZ80 User's Manual[1]
- dastaZ80 Programmer's Reference Guide[2]
- dzOS Github Repository[3]

Contents

1 Boards and Case

The final aim of dastaZ80 is to be a single board computer, but at the moment I am making small *module boards* so that I can test and troubleshoot independently. Plus it allows me to easily upgrade and test (e.g. when I changed the serial board from a MC68B50 ACIA to a Zilog SIO/2).

1.1 Main board

This board is the heart of the computer, and contains the CPU chip, the clock circuit, the reset circuit, the ROM chip, the RAM chip, the I/O decoding logic and the memory decoding logic.

1.1.1 CPU

The CPU is a Zilog Z80 (Z0840006PSC) NMOS 40-pin plastic DIP, rated at 6.17 Mhz, but overclocked to 7.3728 MHz.

The signals */INT*, */BUSREQ*,*/WAIT* and */NMI* are connected to 10K pull-up resistor

1.1.2 Clock circuit

This is the system clock, running at 7.3728 MHz, that drives the CPU and the SIO/2 Channels. It is a very simple circuit consisting of a crystal oscillator and a copule of resistors and ceramic capacitors.

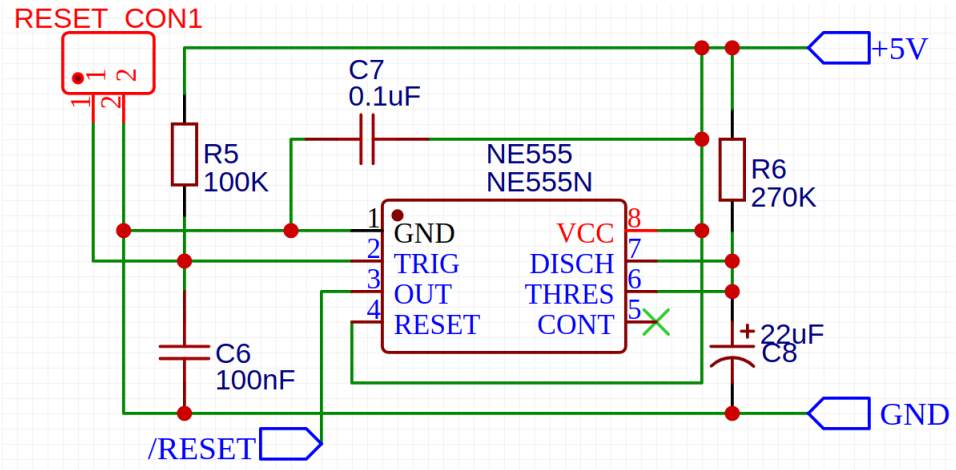
1.1.3 Reset circuit

After power up, the CPU needs to be reset, through the */RESET* signal. When this signal is low for a minimum of three full clock cycles, the CPU resets the interrupt enable flip-flop, clears the Program Counter (`PC`), clears registers `I` and `R`, and sets the interrupt status to Mode 0.

In the dastaZ80, the reset circuit is a bit more complicated than the typical reset circuit found in homebrew computers. The reason is that the VGA output is done with a LILYGO TTGO VGA32 V1.4[1], which needs a few seconds to initialise. So the reset is hold for 6.5 seconds, to allow the initialisation to finish, and then reset the CPU and rest of devices.

Using an NE555 timer running in monostable mode, the */RESET* signal is kept low a number of seconds that can be deduced with the formula: $T = 1.1 * R2 * C2$. (e.g. $1.1 * 270000(270K) * 0.000022(22\mu F) = 6.534 seconds$).

[1] An ESP32 board with a VGA output, that runs FABGL to provide an ANSI terminal.



The *RESET_CON1* is connected to a push-button. R5 and C1 are used as anti-debounce for the button.

1.1.4 ROM chip

The ROM chip is a Winbond W27E512 (64K x 8 bit) EEPROM (Electrically Erasable Programmable Read-Only Memory) 28-pin plastic DIP, mnounted on a ZIF (Zero Insert Force) socket for easy extraction/insertion for programming.

The signal *A15* is connected to *+5V*, therefore is always high, meaning that the start address is `0x8000` and the ROM becomes a 32 KB ROM. This is not the start address that the CPU sees but rather the address where the ROM starts. The CPU will see it as `0x0000`.

As the dzOS is 16 KB in size, I have divided the ROM into two 16 KB ROMs, allowing to execute two different operating systems by selecting the start address (`0x8000` or `0xC000`) via a switch connected to the signal *A14* of the ROM chip.

When the switch is in the lower position, *A14* is connected to *Ground*, thus start address `0x0000` starts at `0x8000` in the ROM. When the switch is in the upper position, *A14* is connected to *+5V*, thus start address is at `0xC000`.

The idea is to be able to use the computer with either dzOS or Digital Research, Inc. CP/M. Of course, the SD will need to be changed for the specific operating system.

1.1.5 RAM chip

The RAM is an AS6C1008 (128K x 8 bit) CMOS SRAM 32-pin plastic DIP.

As the Z80 can only address 65,536 bytes (64 KB), *A16* on this chip is connected to *Ground*, therefore it will always be low, and hence only 65,536 bytes (64 KB) are available.

1.2 Arduino Serial Multi-Device Controller (ASMDC)

An Arduino board acting as a *man-in-the-middle* to dastaZ80 to communicate with different devices like Real-Time Clock, NVRAM, SD card, Floppy Disc Drive, and more.

The dastaZ80's SIO/2 Channel B is connected to an Arduino Mega2560's serial port. The dastaZ80 sends commands (identified by a unique single byte) to the Arduino, which interprets the received command and communicates with the corresponding attached device (e.g. get time from the RTC), and send information back to the dastaZ80.

1.2.1 Floppy Disk Drive (FDD)

This module is connected to original Acorn A3010 FDD (Citizen OSDA-20C).

The available commands (cmd) are:

cmd	Description	(Size) Input	(Size) Output
A0	FDD Status	(0)	(1) see below
A1	FDD Busy	(0)	(1) 0x00=Not Busy 0x01=Busy
A2	Read sector from Floppy Disk	(2) sector_num_lsb sector_num_msb	(512) Sector contents
A3	Write sector into Floppy Disk	(512) Sector contents	(0)
A4	Checks if a disk is in the drive	(0)	(1) 0x00=Disk is in 0xFF=No disk
A5	Checks if a disk is Write Protected	(0)	(1) 0x00=Protected 0xFF=Unprotected
A6	Set drive as Double-density	(0)	(0)
A7	Set drive as High-density	(0)	(0)
A8	Low-level format (nop file sustem)	(0)	(1) Return code (0=success)
AA	Turn FDD motor ON	(0)	(0)
AB	Turn FDD motor OFF	(0)	(0)

A0 (FDD Status)

Tells the status of the FDD.

This command should be called after each command on the **FDD** to check if the command was successful or not. Any value other than `0x00` indicates and error.

- **Lower Nibble** (`0x00` if all OK)
  - **bit 0** = not used
  - **bit 1** = not used
  - **bit 2** = set if last command resulted in error
  - **bit 3** = not used
- **Upper Nibble** (error code)

1.2.2 Micro SD Card Module

This module is connected to the Arduino via SPI.

The available commands (cmd) are:

cmd	Description	(Size) Input	(Size) Output
B0	SD Card status	(0)	(1) see below
B1	SD Card Busy	(0)	(1) 0x00=Not Busy 0x01=Busy
B2	Read sector from Image in SD Card	(2) sector_num_lsb sector_num_msb	(512) Sector contents
B3	Write sector into Image in SD Card	(512) Sector contents	(0)
B4	Close Image File	(0)	(0)
B5	Open Image File	(0)	(0)

B0 (SD Card Status)

Tells the status of the SD Card reader.

This command should be called after each command on the **SD card** to check if the command was successful or not. Any value other than `0x00` indicates and error.

- **Lower Nibble** (`0x00` if all OK)
  - **bit 0** = set if **SD card** was not found

  - **bit 1** = set if image file was not found
  - **bit 2** = set if last command resulted in error
  - **bit 3** = not used
- **Upper Nibble** (number of disk image files found)

1.2.3 Real-Time Clock (RTC) Module

The RTC is a DS3231 module connected to the Arduino via I2C.

The available commands (cmd) are:

cmd	Description	(Size) Input	(Size) Output
C0	Get RTC info	(0)	(1) see below
C1	Check Battery Health	(0)	(1) 0xA0=Healthy 0x00=Dead
C2	Get current Date (in Hexadecimal)	(0)	(9) CCYYMMDDW
C3	Get current Time (in Hexadecimal)	(0)	(6) HHMMSS
C4	Set Date	(7) YYMMDDW	(0)
C5	Set Time	(6) HHMMSS	(0)

C0 (Get RTC info)

Tells information of the configuration of the **RTC** module.

Useful, for example if the time is not kept accurately, to check if the clock is running or not.

- **Byte 1**
  - **bit 0** = set if clock is running
  - **bit 1** = set if clock is in 24 hours mode. Otherwise, clock is in 12 hours mode
  - **bit 2** = set if alarm 1 is ON
  - **bit 3** = set if alarm 2 is ON

1.2.4 NVRAM

This module is part of the Real-Time Clock module.

This module is not currently used by dzOS.

The available commands (cmd) are:

cmd	Description	(Size) Input	(Size) Output
D0	Test NVRAM can be written	(0)	(1) NVRAM capacity (in bytes) or 0xFF if failure
D1	Clear (Set all to zeros) NVRAM	(0)	(0)

1.3 Serial board

The serial board consists of a Zilog SIO/2 (Z84C4208PEG) CMOS 40-pin plastic DIP, rated at 8 MHz, that offers two independent full-duplex channels for data serial communication.

Channel A is used for communication with the Keyboard Interface and the VGA Interface. The Transmit signal (TX) of the Keyboard Interface is connected to the Receive signal (RX) of the Channel A, and the Transmit signal (TX) of the Channel A is connected to the Receive signal (RX) of the VGA Interface.

The type of implementation allows for easy replacement of the keyboard and the screen output with any other serial terminal.

Channel B is not used at the moment, but the aim is to add a MAX232 and a RS-232 9-pin male connector to offer serial communication with other devices.

Both channels are initialised for: 115,200 bps, 8N1

1.4 Keyboard Interface

The keyboard is an Acorn Archimedes A3010. The keyboard matrix is connected via its ribbon cable to a Teensy++ 2.0, which reads the status of the keys and sends the keystrokes via the Teensy serial pin to the SIO/2 Channel A.

A debouncing delay is applied to avoid the mechanical bouncing effect of keyboards, and keys are sent at a configurable (in the controller code) interval for as long as the key is pressed down.

The interface sends ASCII values for all printable keys (i.e. alphabetical A to Z, numerical 0 to 9, and symbols lile !, @, %, etc.). The rest of the keys are interpreted as special keys and special codes are sent.

1.4.1 Special keys

Two special keys have been implemented for dastaZ80:

- **Break/Pause**: At the press of this key, the screen is cleared.

- **ScrollLock**: It is possible to connect the Keyboard Controller to a modern PC via USB cable, at the same time that is acting as dastaZ80 keyboard. To avoid that characters are typed in both computers at the same time, the ScrollLock key allows to swicth between sending only to Serial (dastaZ80) or sending only to USB. When the key LED is illuminated, USB is active.

1.5 Dual Video Output

1.5.1 VGA Output

The VGA output is achieved in a rather simple manner; the SIO/2 Channel A sends its output to a LILYGO TTGO VGA32 V1.4, which runs a very simple ANSI 16 colours terminal emulator using the FabGL library.

This output is referred in the dastaZ80 manuals as *High Resolution* and it is meant to be used as a display for applications.

1.5.2 Composite Output

The Composite output is achieved with a Texas Instruments TMS9918A[2], which provides an NTSC[3] signal.

This output is referred in the dastaZ80 manuals as *Low Resolution* and it is meant to be used as a display for applications' graphics and video games.

Therefore, as the focus for this output should be graphics rather than text, the Operating System does not contain fonts (Patterns) for text. Also, because the OS only uses this output in *Graphics II Bit-mapped Mode*, this is the only mode that can be initialised via a BIOS function call. It is up for each application to initialise the video controller to the desired Mode[4] and to copy the Patterns and Sprites to the TMS9918A's **VRAM**.

For technical information on the TMS9918A, refer to the Texas Instruments' *Video Display Processors Programmer's Guide*[4], or to the article *High-Resolution Sprite-Oriented Color Graphics*[5] published on BYTE magazine.

[2]The TMS9918 and its variants were used in the ColecoVision, CreatiVision, Memotech MTX, MSX, SG-1000/SC-3000, Spectravideo, Sord M5, Tatung Einstein, Texas Instruments TI-99/4, Casio PV-2000, and Tomy Tutor.

[3]National Television System Committee (NTSC) is the standard for analog television used mainly in USA, Japan and some parts of South America

[4]The TMS9918A has four documented modes: Mode 0 (Text), Mode 1 (Graphics I), Mode 2 (Graphics 2) and Mode 3 (Multicolour)

1.6 Programmable Sound Generator (PSG)

The sound generation is provided by a General Instrument AY-3-8912 [5], which is the same as the original AY-3-8910 but with only one parallel port instead of two. This fact does not affect the sound generation.

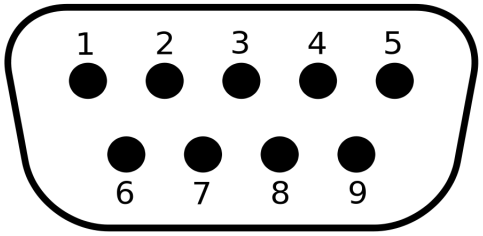
1.7 Dual Digital Joystick Port

The Dual Digital Joystick Port consist of two DE-9 Male connectors for connection of one or two *Atari joystick ports*[6].

Compatible joysticks are those used on Atari 800, Atari VCS, Atari ST, VIC-20, C64, Amiga and ZX Spectrum. Other joysticks, like the ones for the Amstrad CPC and MSX, changed the +5V and GND pins, so it MUST not be used.

1.7.1 Pinout

Seen from the back of the computer case.



Pin	Function		Pin	Function
1	Up		6	Fire
2	Down		7	+5V
3	Left		8	GND
4	Right		9	Not connected
5	Not connected			

1.8 Backplane

The backplane is where all the other boards are connected to. It consists of a single board with six double row 80-pin female connectors at 90 degrees angle.

The connectors drive all CPU signals and the power (+5V and Ground).

[5] The AY-3-8912 and its variants were used in numerous Arcade machines and in home computers like the Amstrad CPC, Atari ST, MSX, ZX Spectrum and others.

[6] Originally introduced on the Atari 2600 home video game console in 1977, became the *de facto* standard in the 1980s.

1.9 Power Supply

The whole computer is powered via an external 5V/4A AC adaptor, which supplies voltage to the different internal parts of the computer.

The different current consumptions are:

- dastaZ80 boards: 133mA (nominal)
- TMS9918A (Composite video output): 200mA (nominal)
- LILYGO TTGO VGA32 V1.4 (VGA video output): 30mA (nominal)
- Arduino Mega2560 (ASMDC): 800mA (nominal)
- Teensy++ 2.0 (Keyboard Controller): 35.5mA (nominal)
- Floppy Disk Drive: 2A (max.)

This totals for 3.2A

1.10 Computer Case

All the boards are inside an Acorn Archimedes A3010 all-in-one case that I had as spare.

The keyboard in the Acorn A3010 case is very PC-like, it has 12 function keys and LEDs for Caps Lock, Scroll Lock and Num Lock.

The back of the case offers holes for connectors for: one DB-25 parallel, one DB-9 serial, two DE-9 joysticks, one stereo jack, one VGA, one RF and ON/OFF switch.

Also there is a hole in the left side for a reset button, and a disk drive bay at the right side.

At the moment, only the ON/OFF switch, VGA connector, Composite video RCA output, joystick 1, joystick 2 and reset button are used.

2 I/O Decoding

The Z80 communicates with devices via the Address and Data buses. And then the signals */MREQ* (for memory devices) and *(*/IORQ) (for I/O devices).

To avoid bus contention[7], we need to enable each device one at a time.

dastaZ80 uses a 74HCT138 (3-to-8 Line Decoder) to allow the CPU to enable I/O devices (non-memory devices). The outputs are enabled (low) when /MREQ is high, /IORQ is low and A7 is low (i.e. addresses below 0x80). This configuration gives 16 addresses for each device (e.g. `0x00` to `0x0F` for a device attached to output /Y0), for a total of 8 devices.

A6	A5	A4	74138 output	Addresses	Device
0	0	0	/Y0	`0x00 - 0x0F`	For future use
0	0	1	/Y1	`0x10 - 0x1F`	**TMS9918A (VDP)**
0	1	0	/Y2	`0x20 - 0x2F`	**AY-3-8912 (PSG)**
0	1	1	/Y3	`0x30 - 0x3F`	**ROM Page** (`0x38`)
1	0	0	/Y4	`0x40 - 0x4F`	**Dual Joystick Port**
1	0	1	/Y5	`0x50 - 0x5F`	For future use
1	1	0	/Y6	`0x60 - 0x6F`	For future use
1	1	1	/Y7	`0x70 - 0x7F`	For future use

[7] Bus contention occurs when all devices communicate directly with each other through a single shared channel (Address and Data buses), and more than one device attempts to place values on the channel at the same time.

3 Memory Decoding

As dastaZ80 has a ROM chip and a RAM chip, the CPU has to decide from which chip to read and to which chip to write. Actually, write operations are only applicable to the RAM chip.

Also, dzOS[8] resides in the ROM chip, but at boot the entire OS is copied into RAM so that it can be modified by the user. Hence, dastaZ80 needs a way to disable the ROM chip after the copy has finished.

3.1 Extra signals

To make the logic even more tight, dastaZ80 uses a 74HCT32 (Quad 2-Input or Gates) to create three new signals from signals comming from the Z80 CPU:

- */WR* or */MREQ* = */MEMWR*
- */RD* or */MREQ* = */MEMRD*
- */WR* or */MEMREG_SEL*[9] = */MEMREG*

3.2 ROM Paging

The ROM paging (disable the ROM chip and enable the RAM chip for all memory related operations) is done with a 74HCT273 (Octal D-type Flip-Flop with Clear) acting as a external register, where the output 1 of this chip is defined as */ROMPAGE*.

When a 0 or a 1 is put into the register's input D0 and */MEMREG* becomes high, then */ROMPAGE* signal becomes low or high.

This 74HCT273 is identified in the circuit as *MEMREGCFG*.

3.3 Putting all together

At power-on or after reset, the 74HCT273 is reset and all outputs are set to low, therefore */ROMPAGE* becomes low. From now on, when there is a memory read (*/MEMRD* low) for an address lower than 0x4000 (*A14* and *A15* low), the ROM chip is enabled (*/ROM_CE low*). When the address is equal or higher than 0x4000 (*A14* or *A15* high), the RAM chip is enabled (*/ROM_CE* high).

When there is an IO write (*/IORQ* and */WR* low) to the any address from 0x30 to 0x3F, */MEMREG_SEL* becomes low and therefore */MEMREG* be-

[8] dzOS is the Operating System of dastaZ80.
[9] Output /Y3 from the I/O Decoder 74HCT138

comes low and *MEMREG*[10] becomes high. This (*MEMREG* high) triggers the Flip-Flop on the 74HCT273, which takes whatever is at the inputs (CPU Data bus: *D0..D7*) and latches it to the outputs. Thus, if *D0* contains a 1, */ROMPAGE* will go high and stay like that until *MEMREG* goes high again. From now on, when there is a memory read (*/MEMRD* low) */ROM_CE* will be high independently of the address (*A14* or *A15*, low or high), thus only enabling the RAM chip.

[10] *MEMREG* is an inverted signal of */MEMREG* done via a 74HCT04

4 Appendixes

4.1 History (Timeline) of dastaZ80

These are the experiences of a guy that had very basic knowledge about electronics[11] but decided to build his own computer from scratch and make an operating system and some software for it.

- **May 2012**
  - I decided to go ahead with my idea of programming an operating system, and decided to do it for a computer that I'll build and call *DastaZ80*.
  - I start first designs (in paper) on how to connect CPU to ROM and RAM.
- **June 2012**
  - First test with a Z80 on a breadboard.
- **July 2012**
  - I've put together a breadboard with CPU, ROM, Clock, SIO/0 and MAX232, and connected it to a PC through the serial port. Nothing appears on the terminal emulator on the PC. It could be a dozen of things :-(
- **August-December 2012**
  - Frustrating period trying to make the computer work with the serial port and not having any luck.
- **January 2013**
  - I've built different test circuits. None worked.
- **February 2013**
  - I've built a debug board (consisting just of six 7-segment LED displays) and connected it to simple circuit (consisting of CPU and ROM). I've programmed the ROM with a simple counter program (counting from 0 to 15 and then from 15 to 0). The result is quite disappointing; it seems the Z80 just reads all instructions, but it doesn't execute them, because I've noticed that it doesn't do the loops but just reads from 0000h to 00012h (i.e. from start to *HALT*). I've tested the CPU on my Amstrad CPC 6128 and

[11] When I started this project, my only knowledge about electronics was; Ohm's and Kirchhoff's laws, and some basic understanding on how capacitors, diodes and resistors work.

it works, so I discard the possibility that the CPU doesn't work. No idea what's happening, and even worse, no idea what to do.

  - After some more tests, I discovered something; I obtain different results when I touch/reseat the cables on the breadboard! It seems either the cables or the breadboard itself are making false connections. This could be the root of the problems I had up until now. I have sockets for the Z80 and AT28C64B. I'm going to start soldering a basic circuit and test again.

- **September 2014**
  - Relocated to another country. Project is halted.
- **May 2019**
  - I re-start the project, using an software emulator I made as a starting point, so that I can focus on developing the OS.
  - The Serial Interface with an MC6850B ACIA is now working and I can connect to the dastaZ80. The OS is working.
- **July 2020**
  - DZOS booting and CLI accepting first commands.
  - The implementation of the FAT file system has taken most of my time, and still I don't have it fully working. It was not fun. I'll take a break.
- **May 2022**
  - I re-start the project (again). Decided to implement a simpler file system that I call dastaZ80 File System (DZFS).
- **June 2022**
  - DZFS is working. I can format and catalogue the disk, load files into memory and run them.
- **July 2022**
  - Thanks to the VGA32 I have VGA output, and thanks to a modified version of my Teensy++ 2.0 based keyboard controller for Acorn Archimedes A3010 keyboard I have a keyboard working.
  - I've also designed a backplane to fit temporarily my boards inside the A3010 computer case until I make a single board. I designed the backplane with EasyEDA and ordered the PCBs at JLCPCB. I'm waiting for the PCBs to arrive.
- **August 2022**

  - *ROM Paging* implemented. DZOS gets copied to RAM and then ROM is disabled.
  - Backplane boards arrived and works perfectly. Only to take in consideration that the CompactFlash board needs to be very close to the CPU to work properly. I'll research about buffering bus lines to see if it works while putting the CF card a bit far away. I need this, because I want to have the CF card accessible at the back of the case, so that CF cards can be exchanged.

- **September 2022**
  - Mark I fully assembled and working.
  - I've made a design with EasyEDA for a PCB that contains CPU, RAM, ROM, Clock and Reset. It's called the *Main Board*. The idea is to free two slots of the Mark I backplane, so that I can fit more boards in the future, like for example RTC, VDP. Ordered it at JLCPCB. I'm waiting for the PCBs to arrive.

- **October 2022**
  - Soldered the components of the Main Board. Some issues on the design found, but I managed to make it work with some workarounds.

- **November 2022**
  - I've decided to go away from building more (for now) boards. Prices of components plus PCBs are too expensive for my taste. I've build an Arduino controller (I've called it *Arduino Serial Multi-Device Controller (ASMDC)*) that controls the RTC and the SD Card (no more CF Card). It communicates with the CPU via serial on the SIO/2.

- **December 2022**
  - I've added the original Acorn A3010 FDD to ASMDC. It works.
  - Starting the Mark II by adding a Texas Instruments TMS9918A (VDP) for graphics video output.
  - I designed a board for the TMS9918A that can be tested first with Arduino and then easily, just two jumpers, modified to work with dastaZ80.
  - Testing the VDP board with Arduino was successful:
    * At first I was getting good results in text mode, but in graphics mode I had no sprites and text lines were duplicated in

pairs every odd line. For several hours I checked and double-checked and triple-checked my circuit and Arduino code. I couldn't find the problem. Finally, reading the VDP manual I noticed that the nominal current need of the VDP is about 200mA, which I'm not sure the Arduino Uno can deliver. I plugged the board to a DC-DC converter 5V/2A and all was working. Sprites and correct text lines. But a huge rippled image on the screen.

    * Testing with another power supply brought crispy image. It seems those DC-DC step down converters I bought have a big voltage ripple on the output.

    * Also, image on CRT was B/N, due to signal being NTSC. But with the addition of a NTSC-to-PAL converter all looks fine.

  - Testing the VDP board with the dastaz80 was successful, though there are some artifacts on the screen. Looking at the contents of the **VRAM** with the *vramdump* program, I can see that in the Colour Table some bytes have the value *0xF1*, which is the value I initialised the Colour Table with. So it seems some skipping is happening. I added *NOP* instructions, but no change so far. And why only happens with the Colour Table?

  - Found that the VDP needs 2 µs to read or write a byte to its RAM. A *NOP* instruction at 7.3728 Mhz only takes 0.54 µs, so I need at least four *NOP*s. Instead I went for creating a subroutine, that with its instructions plus the *call* takes 6.24 µs

  - Last day of the year! I built and tested successfully a circuit for the Dual Joystick Port.

- **January 2023**

  - I've spent the first days of the year making programs to test the **VDP**. Changing screen modes, loading fonts, writing text to the screen, and displaying images all successful.

  - I've also designed a new circuit that contains the Dual Joystick Port and the AY-3-8912 PSG. All tests were successful.

  - I've also installed at the back of the case a 3.5mm jack to 3 RCA Audio/Video cable. The Composite video and the stereo audio are connected to this jack, to be used with a cable like those used for Raspberry Pi.

- **July 2023**

  - After a few months of pause, I continue (sporadically though) with development of software (e.g. text editor, testing PSG, adding more commands to MS BASIC, and more).

4.2 Acknowledgements and Thanks

- **Albert P. Malvino**, for his books *Electronic Principles*[6] and *Digital Computer Electronics* [7].
- **M. Morris Mano**, for his book *Computer System Architecture*[8].
- **Steve Ciarcia**, for his awesome and inspirational book *Build your own Z80 computer*[9], which inspired me into starting this project.
- **Grant Searle**, for his great *My simple Z80 design* [10], which also inspired me to do this project. For his work on the NASCOM MS BASIC. And for having the patience to answer all my questions.
- **Doctor Volt**, for his *TMS9918 Arduino Library* [11], which allowed me to test my board and finally understand how programming the VDP works.
- **David Hansel**, for his *ArduinoFDC Library* [12], which allowed me to add Floppy Disk Drive support.
- **Zilog, Inc.**, for creating the Z80 microprocessor.
- **Gary A. Kildall**, for his CP/M, which inspired my own operating system DZOS.
- **My father**, for buying my first computer, an Amstrad CPC 464, and for teaching me the basics of Z80 assembly language.

4.3 Future Improvements

These are some ideas I have for improvements:

- **Cartridge Port**: allowing almost instantaneously load and access to programs stored in EEPROMs.

References

[1] David Asta. *dastaZ80 User's Manual*, 2022.

[2] David Asta. *dastaZ80 Programmer's Reference Guide*, 2022.

[3] David Asta. dzos github repository. https://github.com/dasta400/dzOS, 2022.

[4] Texas Instruments. *Video Display Processors Programmer's Guide*.

[5] Steve Ciarcia. High-resolution sprite-oriented color graphics. *BYTE, the small systems journal*, 7(8):57–72, 1982.

[6] Albert P. Malvino. *Electronic Principles, 6th Edition*. Glencoe/MacGraw-Hill, 1999.

[7] Albert P. Malvino and Jerald A. Brown. *Digital Computer Electronics, Third Edition*. Glencoe/MacGraw-Hill, 1997.

[8] M. Morris Mano. *Computer System Architecture*. Prentice-Hall, Inc., 1976.

[9] Steve Ciarcia. *Build Your Own Z80 Computer, Design Guidelines and Application Notes*. BYTE Books/MacGraw-Hill, 1981.

[10] Grant Searle. My simple z80 design. http://www.searle.wales/.

[11] Doctor Volt. Tms9918 arduino library. https://github.com/michalin/TMS9918_Arduino.

[12] David hansel. Arduinofdc library. https://github.com/dhansel/ArduinoFDC.