dastaZ80 Mark I
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

Hardware is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License**

http://creativecommons.org/licenses/by-sa/4.0/

Software is licensed under **The MIT License**

https://opensource.org/licenses/MIT

Documentation is licensed under the **Creative Commons Attribution-ShareAlike 4.0 International License**

http://creativecommons.org/licenses/by-sa/4.0/

©2022 David Asta

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 80 column VGA output is referred as **CONSOLE**.

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

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

Related Documentation

dastaZ80 User's Manual
dastaZ80 Programmer's Reference Guide
https://github.com/dasta400/dzOS

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.

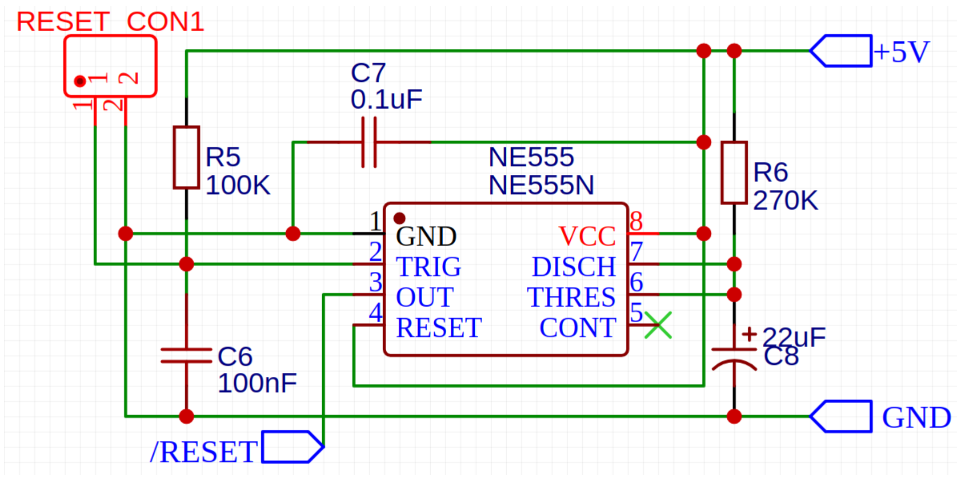
1.1.3 Reset circuit

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 VGA Interface

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.

1.6 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).

1.7 Power Supply

The whole computer is powered via an external 12V/4A AC adaptor, which is connected to five LM25961 DC-DC Step Down Buck Converter with an output of 5V 2.5A.

Each LM25961 supplies independent voltage to: the backplane, the VGA32, the Teensy++ 2.0 (Keyboard controller), the Arduino Mega2560 (ASMDC) and the FDD.

1.8 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 DB-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 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[2], 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`	For future use
0	1	0	/Y2	`0x20 - 0x2F`	For future use
0	1	1	/Y3	`0x30 - 0x3F`	**ROM** Page (`0x38`)
1	0	0	/Y4	`0x40 - 0x4F`	For future use
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

[2]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[3] 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*[4] = */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-

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

comes low and *MEMREG*[5] 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.

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

4 Future Improvements

These are some ideas I have for improvements:

- **Dual video output**: the idea is to have the current VGA output as *High Resolution* output, for high quality 80 column usage, and then add a TMS9918A VDP for *Low Resolution* 40 column graphics (e.g. games).
- **Sound Interface**: stereo sound output with an AY-3-8912.
- **Dual Digital Joystick Port**: Allowing connection of two Commodore 64 compatible joysticks.
- **Cartridge Port**: allowing almost instantaneously load and access to programs stored in EEPROMs.