Z80-Retro! Manual

David Latham, K3130440 (Tim)

2024-01-02

Z80-Retro-Manual

So you've decided to build the Z80-Retro Single Board Computer! Welcome to the club of retro enthusiasts.

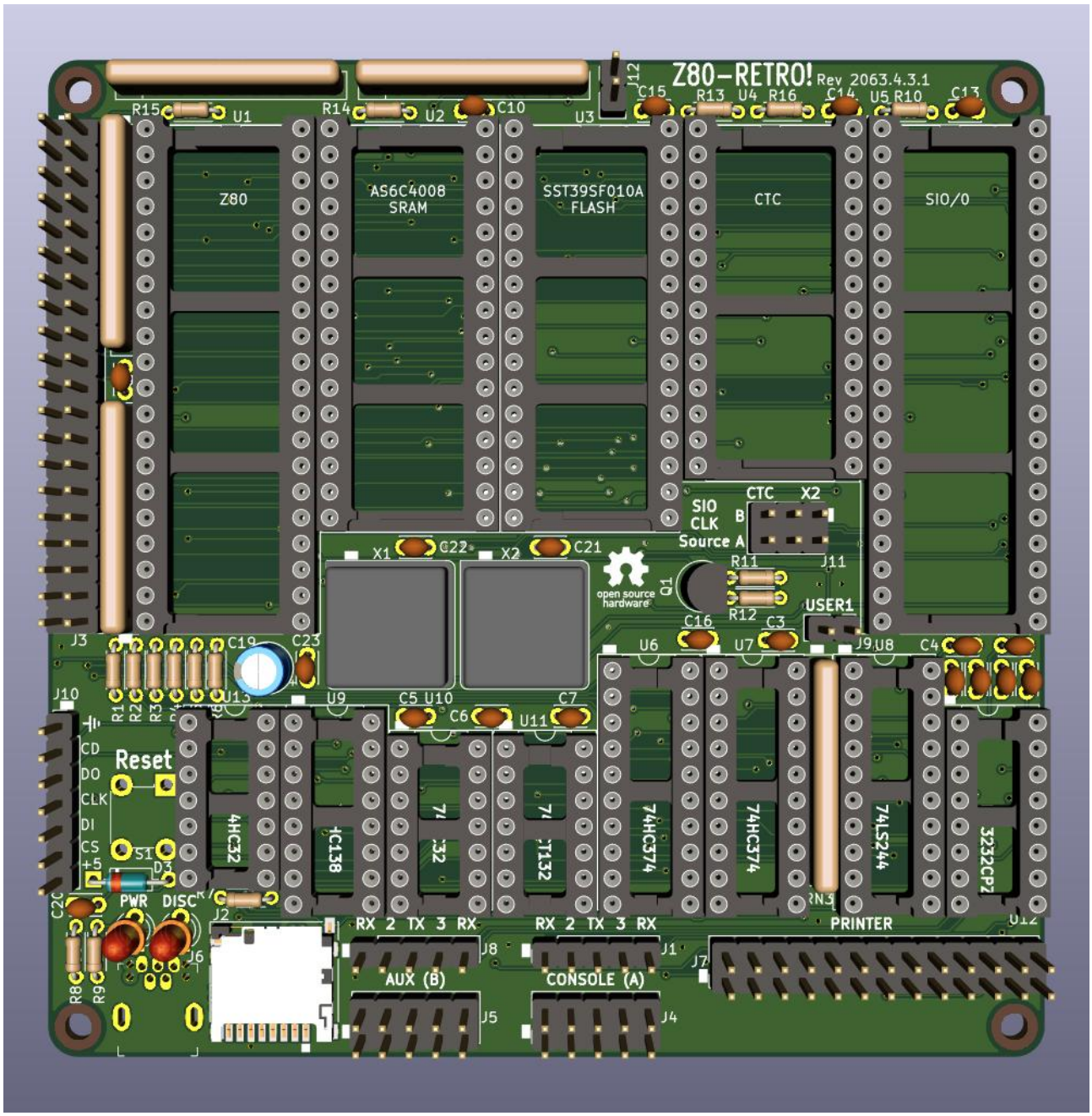

Figure 1: 3D Render of the Z80-Retro CPU Board

This document is intended as a guide to building and setting up your Retro. It is complimentary to the series of YouTube videos that probably brought you here in the first place.

- Z80-Retro GitHub Organization
- Z80-Retro YouTube Playlist

The document is split into different sections. It is recommended to read through all the documents in the order provided below at least once before starting the project. You might need to refer back to this guide multiple times so go ahead and bookmark this page right now.

Sections

- Preface
- General
  - Z80-Retro Architecture
  - Z80-Retro Build
  - Z80-Retro Memory
- Software
  - Z80-Retro Development Environment
  - Z80-Retro Firmware
  - Z80-Retro Install CP/M
  - Z80-Retro File Transfer
  - Z80-Retro IO Ports
- Daughter Boards
  - Z80-Retro VDP Build
- Emulation
  - Using the EtchedPixels EmulatorKit
- Git and Collaboration
  - Z80-Retro Org and Git

Document Conventions

- Links:

  Wherever a better resource exists on the internet that provides additional detail, a link will be provided. For the most part, these will be links to the Johns Basement YouTube videos or GitHub repositories.

  eg: *Video Link: Z80 Retro #7 - 512K Bank Selected Memory*

- Commands:

  Commands that you need to enter into a shell terminal or into a terminal session will be given in this format:

  - Shell (for example on a Raspberry Pi)

    ```
    $ make
    ...
    ```

  - Terminal Session (for example, a minicom session into CP/M)

    ```
    A> stat *.*
    ...
    ```

Prerequisite Knowledge

There is some assumed ability required before beginning the project. You will need some soldering skills as well as an understanding of Linux, Bash, Git and Makefiles. While care has been taken to provide Makefiles that automate many of the tasks, you will still benefit from understanding how they work. Especially in cases where your own environment differs from the environment presented in the Johns Basement videos.

Refresh your knowledge in these areas:

- ☐ Add links to resources
  - REQUIRED: Soldering through hole components
  - REQUIRED: Git workflows
  - REQUIRED: Linux and Bash
  - RECOMMENDED: Simple surface mount soldering techniques

Document Status

This is a working document and readers should expect updates from time to time as errors are fixed and refinements made.

Versions will be released according to the Semantic Versioning Standard 2.0.0.

Contributing

If you find any errors or omissions you can raise an Issue on this GitHub repository. Pull requests are also welcome and will be reviewed.

If you do decide to raise a Pull Request, please be mindful of the how the documented is formatted.

- GitHub style Markdown syntax
- Code snippets must be formatted correctly.

Authors

- John Winans https://github.com/johnwinans
- David Latham https://github.com/linuxplayground

Copyright

Preface

Having recently completed a Z80-Retro board, it became apparent that some form of documented manual might be useful for other folks doing the same thing.

The Z80-Retro is a unique retro project in that the level of detail provided by John is outstanding. There are a few other creators who have done similar things.

- Ben Eater - 8-bit CPU on breadboards and 6502 on breadboards.
- James Sharman - Pipelined CPU.
- Fabian Schuiki - Build a superscaler CPU

Perhaps there are more, but none that take you through the journey of designing, building and programming a retro project that runs CP/M with the same careful consideration for the viewer's learning experience as John's Basement.

Before you even have the supplies you need to get started, you really should join the Z80 Retro! Discord Server, introduce yourself, and join the community.

Here are some photos of my build showing the Z80-Retro! with the VDP duaghter board.

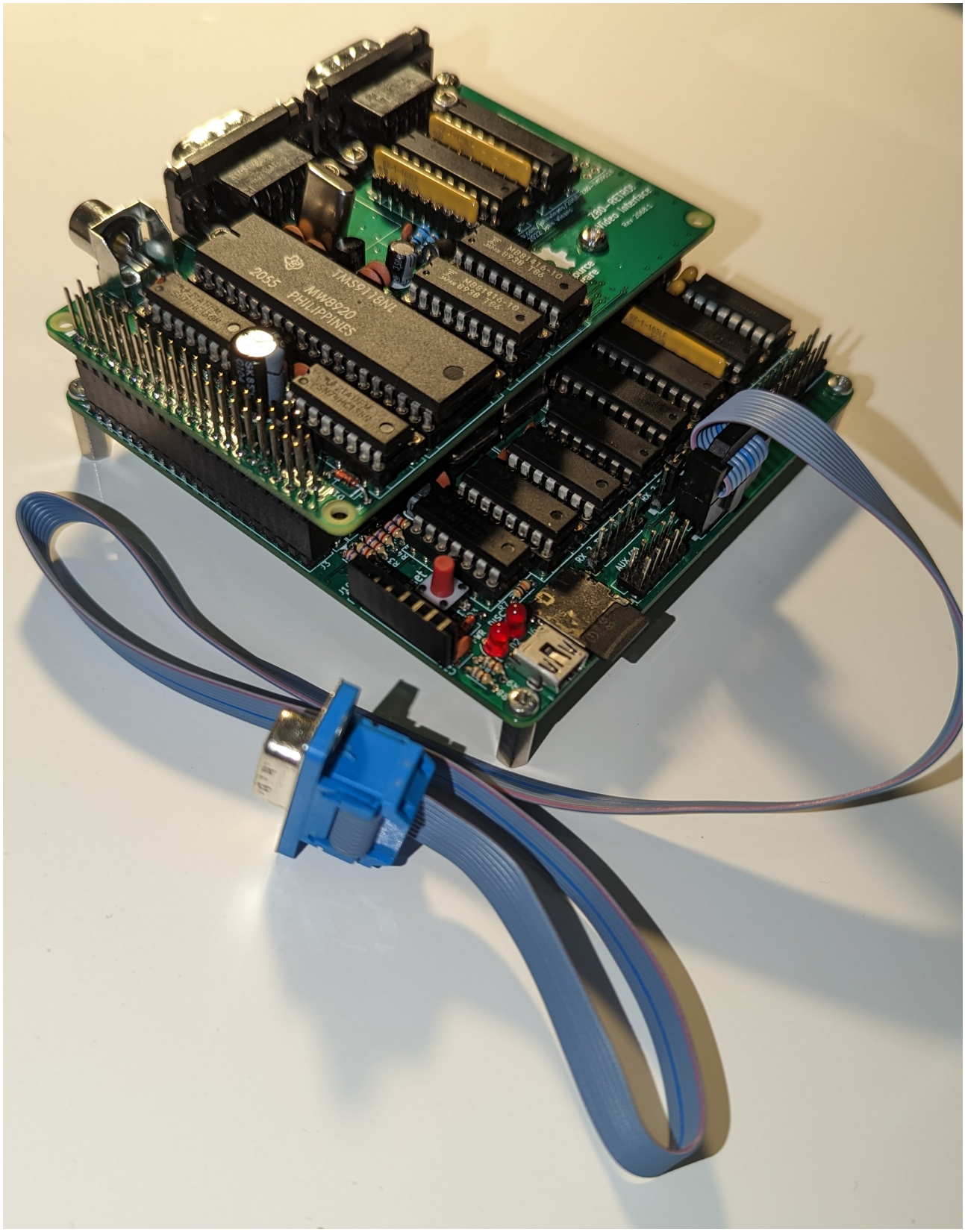

Figure 2: gallery01

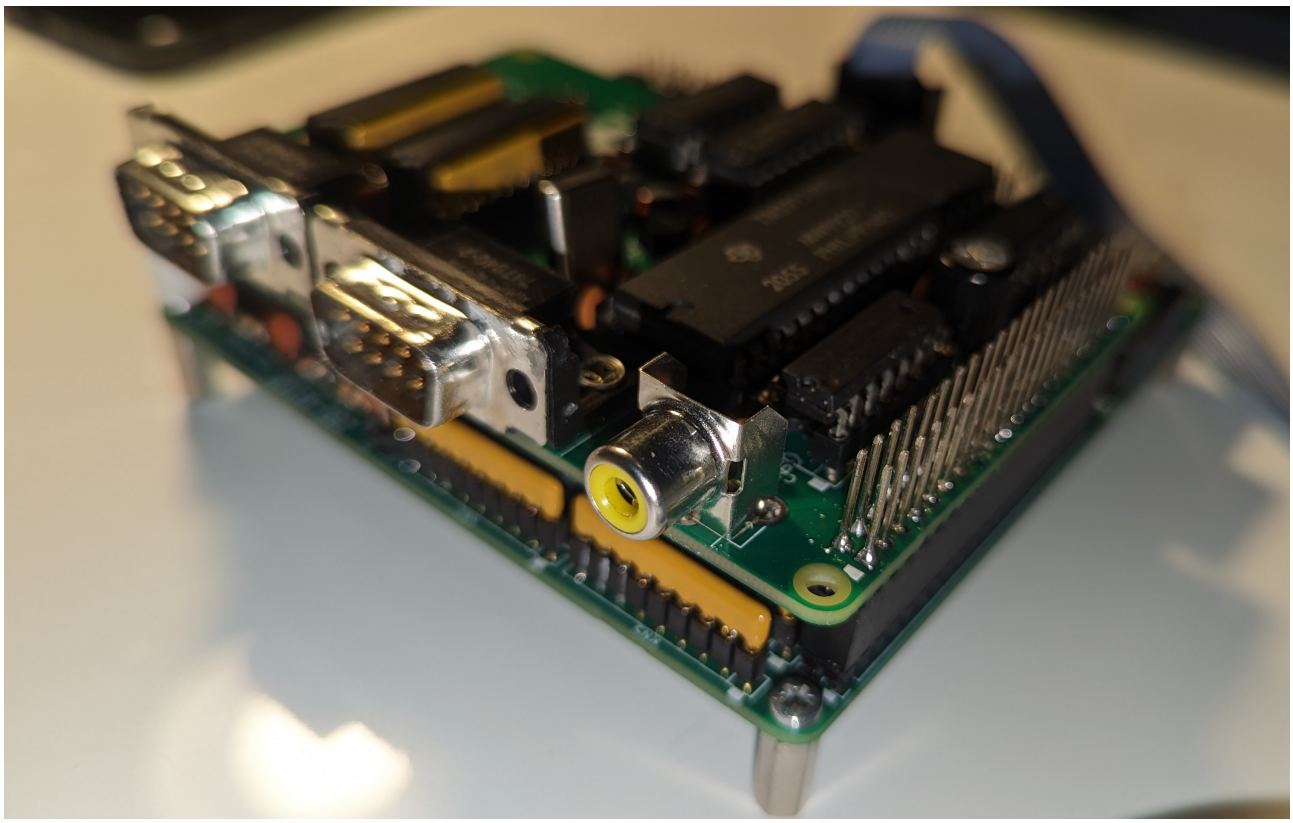

Figure 3: gallery02

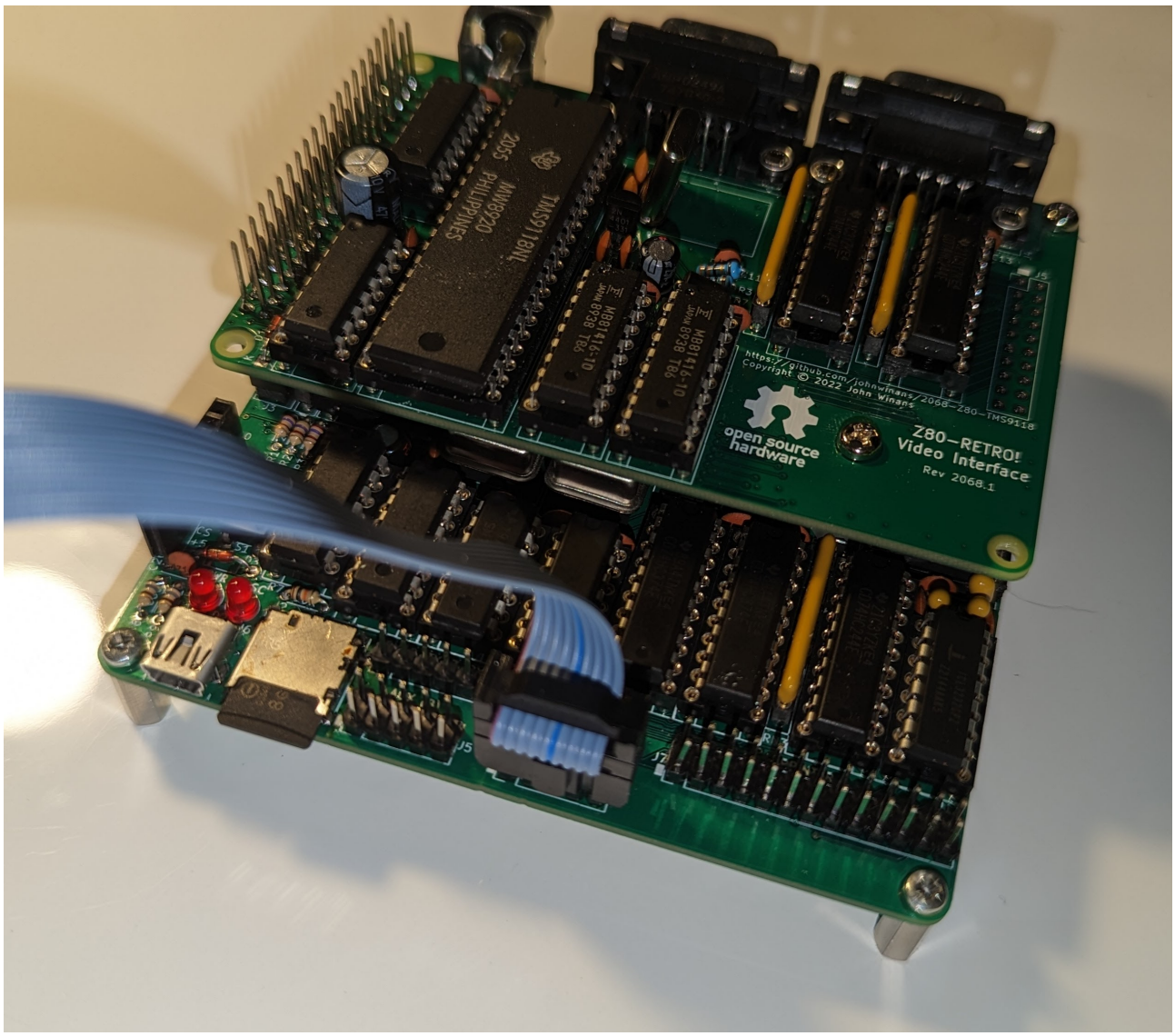

Figure 4: gallery03

Rock On!

Dave Latham - aka (production-dave)

Figure 5: blinky

Z80-Retro Architecture

Back

CPU Board Block Diagram

- *Video Link: Z80 Retro #2 - Block Diagram*

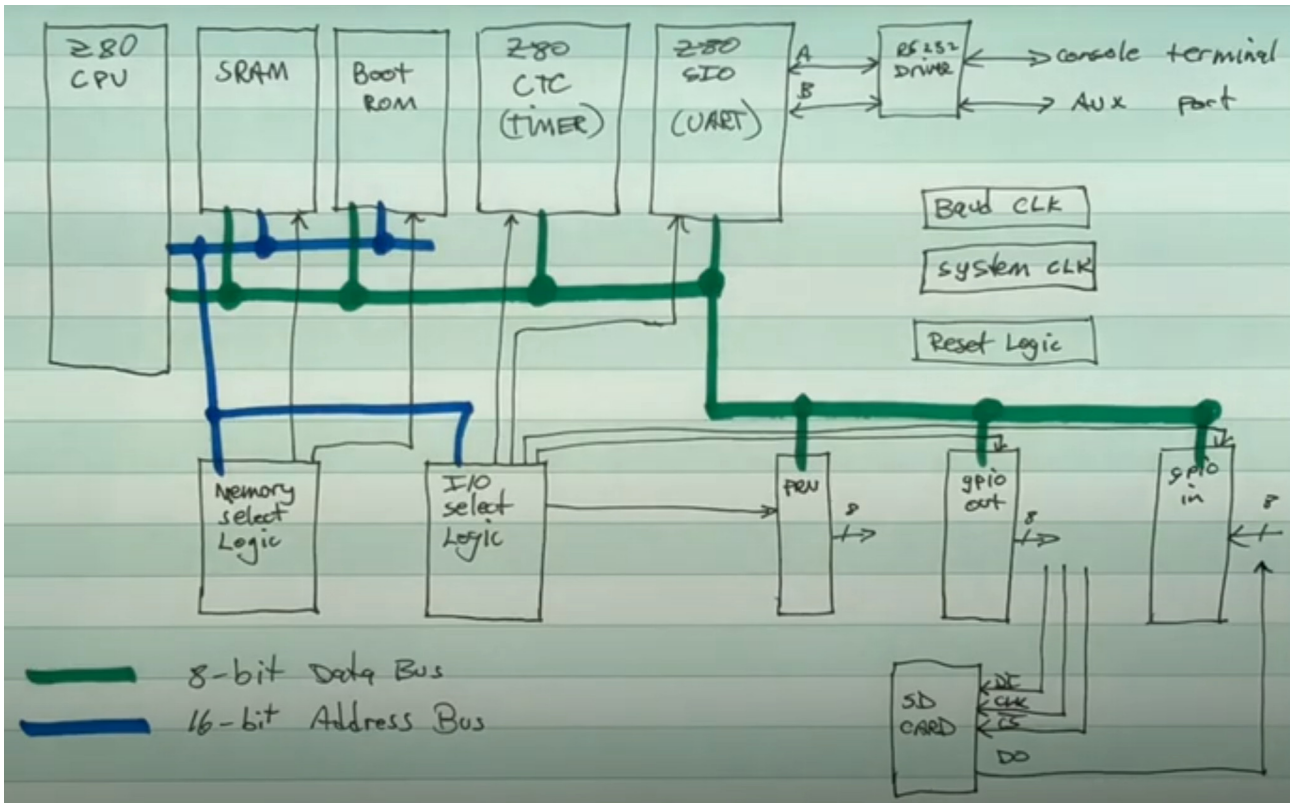

Figure 6: Block Diagram

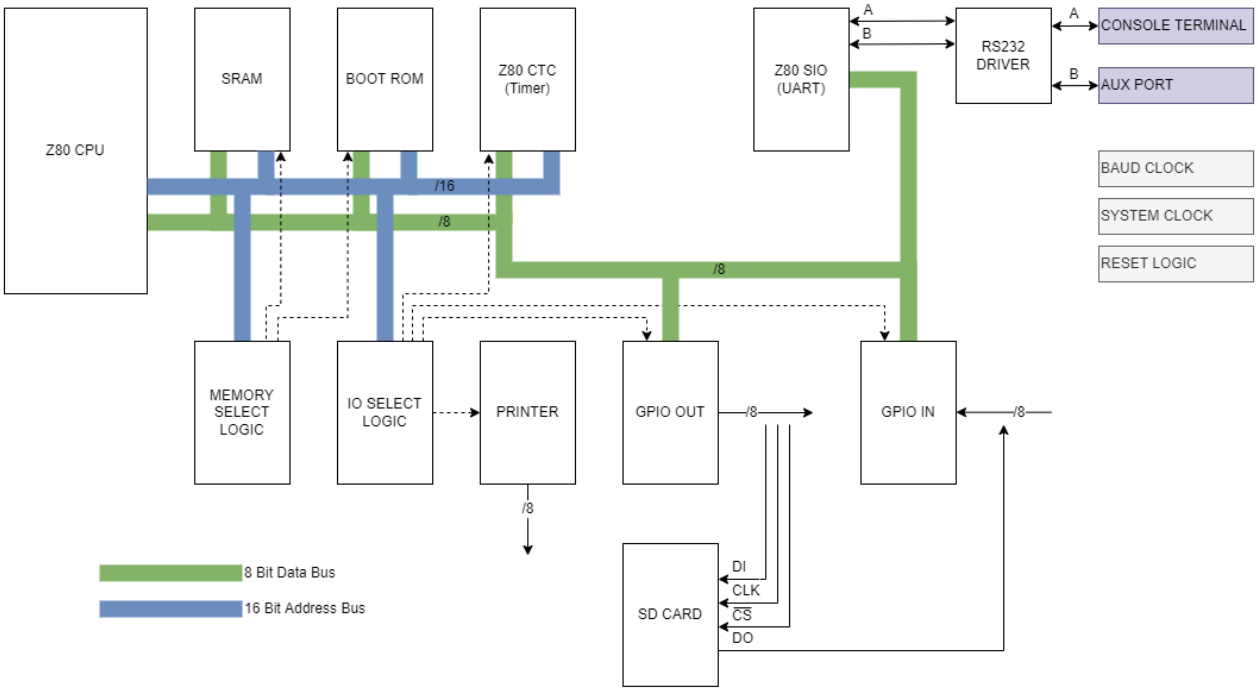

Figure 7: Formatted Block Diagram

The CPU Board is built around the Zilog 8 bit Z80 CPU. It has a 16 bit wide address bus which means it can address a maximum of 64KB of memory.

Design Philosophy

John, the designer, has tried as much as possible to re-create the original Single Board Computer hobby system that was so popular in the early eighties but with a modern twist. It is built almost entirely of modern components. The Z80 Processors are still in production and all the logic ICs are still being made. The only components that are either new old stock or second hand stock are the VDP and the VDP Memory. With the VDP being an *optional* add-on, you could get away with thinking that John has absolutely achieved this goal.

Components

- CPU

  The CPU Board is built around the Zilog 8 bit Z80 CPU. It has a 16 bit wide address bus which means it can address a maximum of 64kb of memory. It also has a separate IO Bus used to access peripherals.

- RAM

  512KB SRAM divided into 16 banks of 32KB each.

- FLASH

  128KB FLASH ROM. The CPU boots from the flash ROM, which in turn is responsible for loading the first 32 blocks of storage from the SD Card into the TOP (Bank 15) bank of RAM and then jumping to the first address in that bank.

- CTC

  This a general purpose counter timer. It has a selection of programmable timers that are used to drive the baud rates for serial communication via the CON and AUX RS232 ports.

- SIO

  A dial serial input output device which is connected to the RS232 chip.

- RS232

  Responsible for converting the TTL (5VT) logic signals into the +12VT / -12VT required for a standard RS232 connection.

- IO Address Decoder

  These ICs convert requested IO Addresses into chip enable signlals for the input buffer and the two D-Type latch devices.

- GP IO and Printer Out

  These chips are responsible for interfacing to the SD Card, Printer as well as holding the memory bank latch.

- SD CARD

  Block storage device interfaced over bit-banged SPI via the GPIO system.

- Power

  Power is provided through a USB Type B Connector on the board. This connector only provides Power. There is also an onboard voltage regulator that is used to provide 3.3VT to the SD Card adapter.

- System Clock

  A 10MHz Full Can Oscilator provides the 10MHz signal to the Z80 CPU.

- Baud Rate Clock

  Optionally divided by the CTC to allow for slower baud rates on the SIO. By default, the Z80-Retro runs serial IO at 115200kbps.

- CPU Connector

  This connector is a 40pin header that can be used to interface with the two daughter boards. It provides all the same signals as those on the CPU.

- Reset Circuit

  A momentary push button style switch that brings the RESET signal low while held down. Used to COLD BOOT the CPU, reset the FLASH latch, and start everything up from scratch.

External Input Output Interfaces

The Z80-Retro has 3 primary interfaces for communication with external devices.

- RS232 CON The main serial interface used to connect to the Z80-Retro over an RS232 serial connection. This is the default serial port to connect to your host PC via an IDC to DB9 adapter cable.

- RS232 AUX Used for auxillary serial communication. The Baud rate of the AUX port can be programatically controlled to support slower rates for odler devies. Typically, one would use this to connect to another serial device or modem.
- Printer A 26pin IDC connector requiring an IDC to DB25 adapter cable to connecting to a line printer.

Daughter Boards

There are two daughter boards available for the Z80-Retro.

Raspberry Pi Flash Programmer

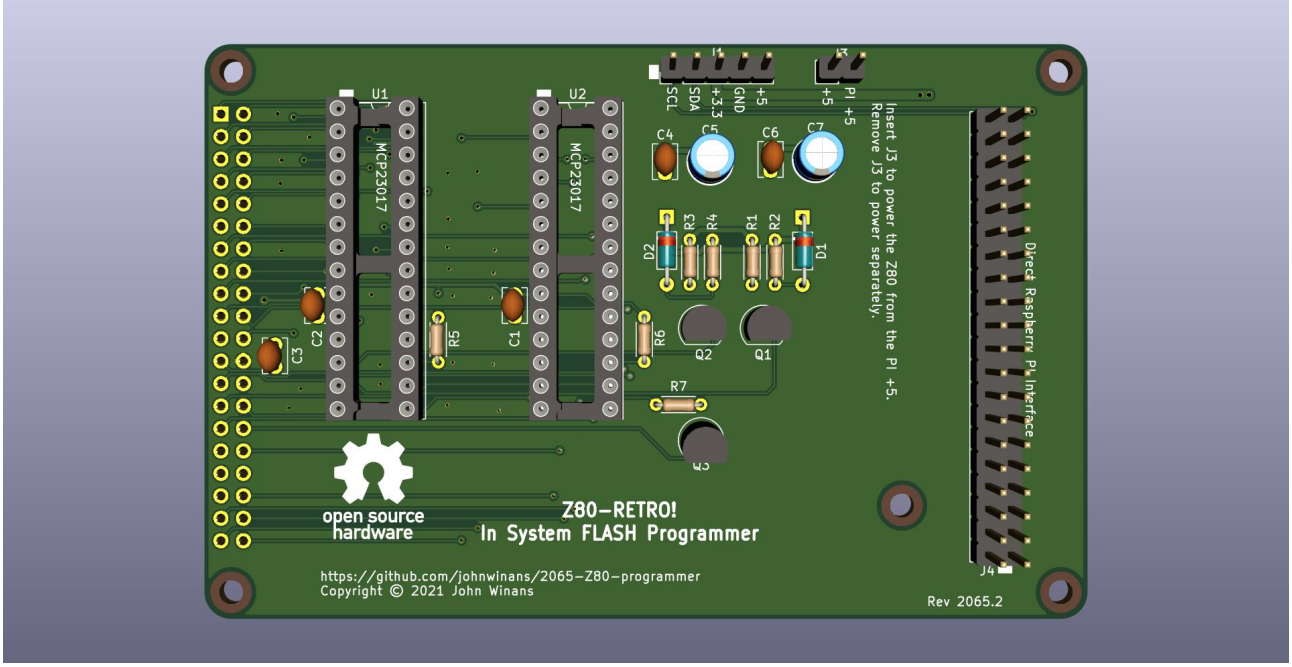

Figure 8: Z80-Retro Programmer Board

This optional programmer is designed to interface directly to the Z80-Retro CPU Board and a standard Raspberry Pi Header. When this is connected and you run the rquired tools, it will take direct control of the address and databuses by asserting the BUSREQ signal to the CPU. Once it has control of the busses, it will program the 128kb FLASH ROM chip.

It is also possible to program the FLASH ROM directly using another kind of programmer that supports the specific FLASH ROM device.

The programmer has on board voltage level shifting to protect the 3.3VT pins on the Raspberry Pi.

TMS9118 VDP Graphics card

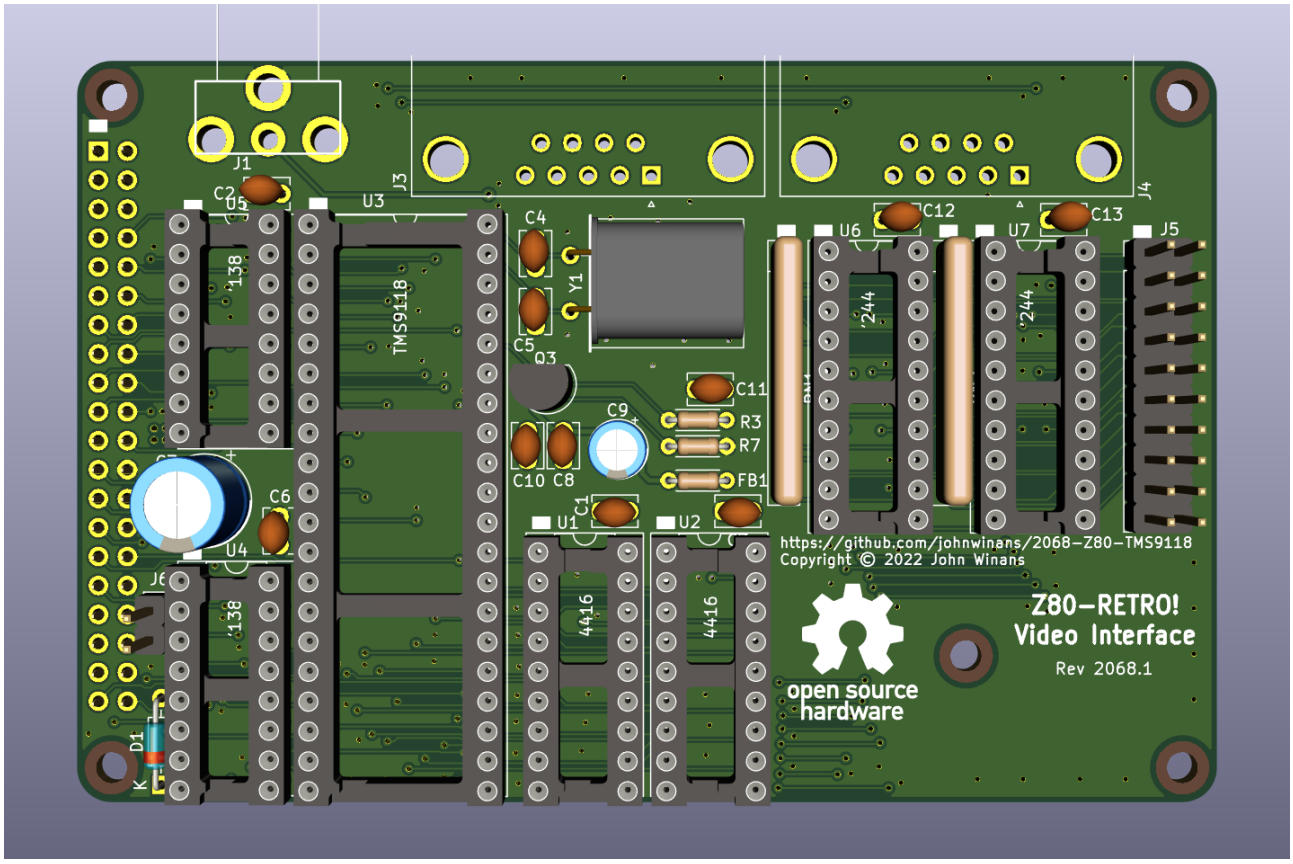

Figure 9: Z80-Retro VDP Board

A graphics card for the Retro outputting a composite NTSC signal. It has its own video memory and can be controlled by the CPU over the IO Bus.

Operating System

The Z80-Retro is designed to work with CP/M version 2.2. There are a range of software options for CP/M 2.2 that can be loaded on to the SD Card and will work.

CP/M is designed as a serial only operating system and works very well on the Z80 platform. It is, therefore, the natural choice of OS.

Z80-Retro Hardware Build Guide

Back

This document is not a “how to solder things” document nor is it going to get into the intricacies of surface mount soldering techniques.

That said, here are some tips:

- Videos:
  - *Video Link: Z80 Retro #33 - v4rc1 Full Build and Test pt. 1*
  - *Video Link: Z80 Retro #34 - v4rc1 Full Build and Test pt. 2*
- Use good quality, machined sockets for the full can oscillators.
- Watch out when ordering the resistors. They need to be the 1/8 watt type. They are very small. The more common 1/4 watt type will not fit well on the board. You can make them work, but it’s gonna be messy.
- If you’re struggling with the surface mounted SD Card socket, have a look at the Sparkfun breakout board option.
- Some people even use sockets for the resistor networks. This does raise them up slightly but as resistor networks are more expensive, you might want to consider using a socket for them too.
- Use ceramic MMCC Capacitors for the un-polarised capacitors. An assortment can be purchased from Amazon. Don’t use tantalum capacitors like I did in the RS232 circuit. While they do work, they are polarised so care should be taken to install them correctly and they are sensitive to voltage spikes and the risk of them popping is high. **DON’T USE THEM.** They are censored in the image below.
- Make sure your oscillator cans are oriented correctly. Pin 1 is pointed to by the orange arrows on the image below.
- You can use the cheaper socket types for everything but the oscillator cans. (I used machined sockets because I am a nerd.) It’s not required.

As Built CPU Board

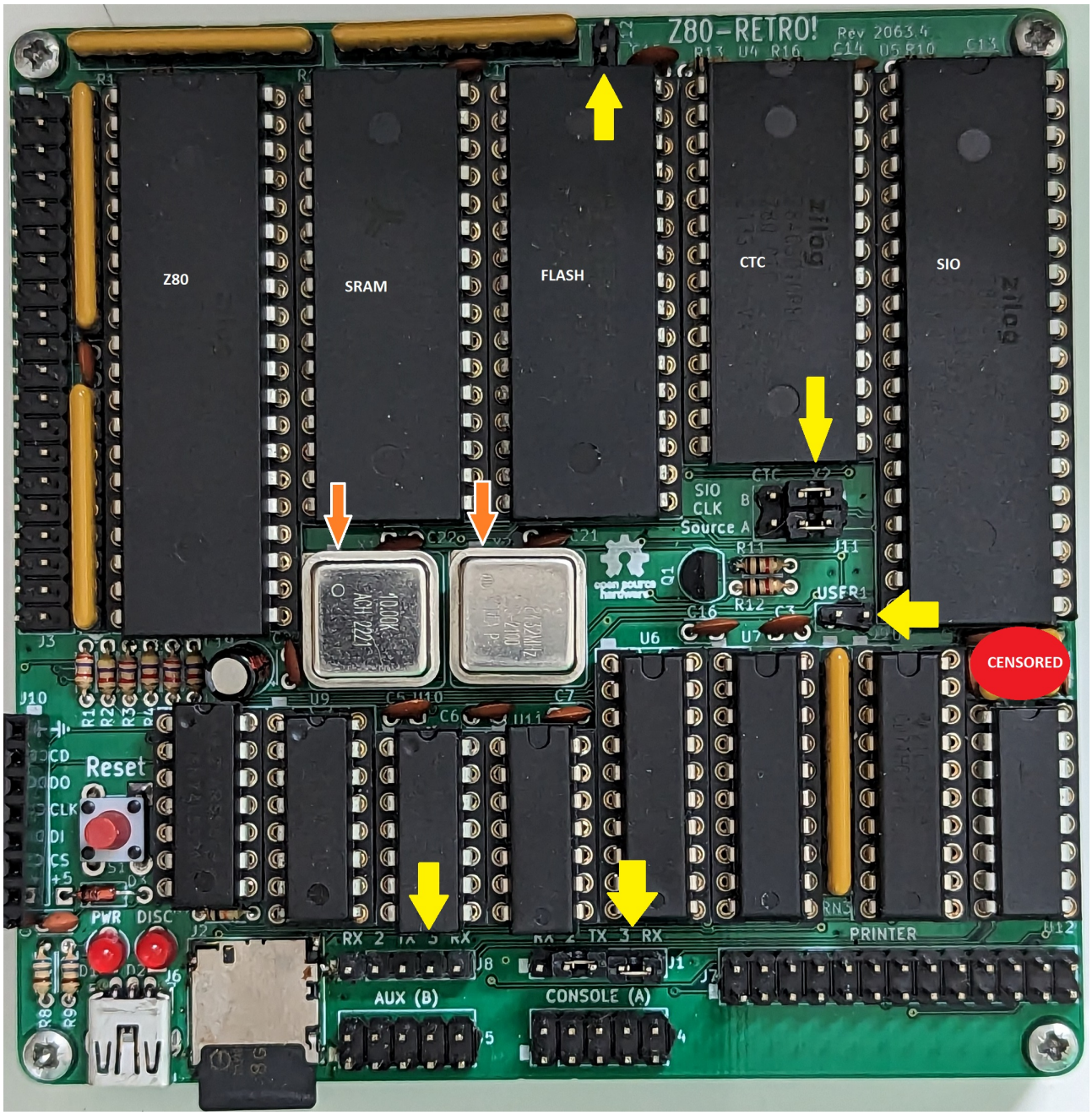

Figure 10: Photo of Z80 Retro As Built

Jumper Settings

There are 5 sets of jumpers on the Z80-Retro marked in Yellow in the above image.

Silk Screen	Function	Default Config
J1 - CONSOLE (A)	Primary serial connection to host PC	TX-PIN2,RX-PIN3
J8 - AUX (B)	Used for auxiliary serial connections	Unpopulated

Silk Screen	Function	Default Config
J11 - SIO/CTC	Set SIO or CTC as clock source for CONSOLE (A) and AUX (B)	Both set to X2
J12	Select a different bank on the Flash.	Unpopulated
J9 - USER1	General user input. Assigned to bit 5 on GP Input.	Unpopulated

Z80-Retro Memory

Back

Z80 Memory

- *Video Link: Z80 Retro #7 - 512K Bank Selected Memory*

As the Z80 processor has a 16bit wide address bus, it can only address 64K of memory. This will include both the boot ROM and the static ram combined.

The Z80 Boot ROM is designed to load the operating system from the SD Card and save it into ram. When that task is complete, it disables the ROM and enters e operating system. Thus, and for the purposes of this section of the manual, any talk of memory will refer to memory in SRAM unless otherwise noted.

Banked Memory Layout

The Z80 Retro SRAM is a 512K SRAM device. It can be divided down into 16 blocks of 32K. The memory select logic is responsible for enabling or disabling banks of RAM such that the CPU has access to a total of 64K of RAM at all times.

BANK	START	END	COMMENT / USAGE
15	0x78000	0x7FFFF	ALWAYS enabled for addresses 0x8000 to 0xFFFF
14	0x70000	0x77FFF	Default Low bank assigned by firmware
13	0x68000	0x6FFFF	
12	0x60000	0x67FFF	
11	0x58000	0x5FFFF	
10	0x50000	0x57FFF	
09	0x48000	0x4FFFF	
08	0x40000	0x47FFF	
07	0x38000	0x3FFFF	
06	0x30000	0x37FFF	
05	0x28000	0x2FFFF	
04	0x20000	0x27FFF	
03	0x18000	0x1FFFF	
02	0x10000	0x17FFF	
01	0x08000	0x0FFFF	
00	0x00000	0x07FFF	

Bank 15 is always enabled whenever the CPU addresses anything in the 0x8000 to 0xFFFF address range. This is also called the HIGH Bank.

The remaining 15 Banks (0-14) can be selected into the LOW Bank or addresses from (0x0000 to 0x7FFF). For example, if bank 05 is selected and the CPU reads from address 0x100 that read will be made against address 0x28000 + 0x100 = 0x28100 in the SRAM device. The CPU has no idea that happened as from its perspective it wants to read 0x100. The memory select logic is deciding which bank of memory in the SRAM range to retrieve that data from.

The BANK Selection Logic

Address lines 15 - 18 on the SRAM chip provide the memory division down to 16x32K banks. These 4 address lines are driven by 4 or gates where each line is ORed together with A15 from the CPU. Thus, if the CPU is requesting data from the high half of memory (0x8000 - 0xFFFF), A15 will be a 1 and a 1 OR x is always = 1. In this case, all 4 lines 15-18 on the SRAM chip will be ones and that maps to the address range 0x78000 - 0x7FFFF (or bank 15)

In the case where the CPU is requesting something from the low half of memory (0x0000 - 0x7FFF) then the other half of those OR Gates will determine which of the SRAM address lines 15-18 are enabled and so a bank is selected.

The GPIO OUT Latch (74HC374) holds the value for the bank selection bits in the 4 most significant bits of its data. Programmers can set the values of these 4 bits to set the current memory bank.

Z80-Retro Development Environment

Back

In order to get set-up on your Retro you will need to compile the firmware, CP/M OS and user applications. These are all z80 assembly to start with and some of the user programs later on are written in C.

These instructions are to get your local development environment set up in preparation for the compilation steps later on.

Linux

The development environment is based on having access to a Linux based operating system. These instructions cover Linux OS Variants based on the Debian package manager such as Ubuntu and Raspian OS.

- ☐ TODO: Add a link to a guide for Windows Users and WSL.

Install Build Dependancies

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential z80asm cpmtools
```

Test that you are able to compile Z80 Assembly code by creating the following source code file and assembling it. If your code assembles and generates a hex file that matches the one listed, you know you are good to go.

```
$ cat hello.asm
bdos:   equ     0x0005          ; BDOS Sys Request
print:  equ     0x09            ; BDOS Print String function

        org     0x0100

        ld      c,print
        ld      de, message
        call    bdos
        ret
message:
        db      0x0d,0x0a,'Hello, World!',0x0a,0x0d,'$'
```

Assemble with z80asm. There should be no errors.

```
$ z80asm -o hello.com --list=hello.lst hello.asm
```

Check that a listing file and a com file were generated.

```
$ ls
hello.asm  hello.com  hello.lst
```

Check the listing file.

```
$ cat hello.lst
# File hello.asm
0000                    bdos:   equ     0x0005          ; BDOS Sys Request
0000                    print:  equ     0x09            ; BDOS Print String
    function
0000
0000                            org     0x0100
0100
0100 0e 09                      ld      c,print
0102 11 09 01                   ld      de, message
0105 cd 05 00                   call    bdos
0108 c9                         ret
0109                    message:
0109 0d 0a .. 0a 0d ..          db      0x0d,0x0a,'Hello, World!',0x0a,0x0d,
    '$'
# End of file hello.asm
011b
```

Check the hex for the com file.

```
$ hexdump -C hello.com
00000000  0e 09 11 09 01 cd 05 00  c9 0d 0a 48 65 6c 6c 6f  |...........
    Hello|
00000010  2c 20 57 6f 72 6c 64 21  0a 0d 24                 |, World!..$|
0000001b
```

If this all looks good, you can choose to delete the 3 hello files now.

Working Directory Set-Up

The best way to organize the code on your computer is to follow the same pattern as John does. Set up a folder beneath which you will run all your projects.

```
$ cd $HOME          # Start out in your home directory.
$ mkdir z80-retro   # Create a folder for your work
$ cd z80-retro      # navigate to the new folder
```

From now on this document will assume that your current working directory will always be the z80-retro directory created in this step.

Clone Repositories

```
$ git clone --recurse-submodules https://github.com/Z80-Retro/2063-Z80-cpm.
    git
$ git clone https://github.com/Z80-Retro/example-filesystem.git
$ git clone https://github.com/Z80-Retro/Z80-Retro-disk-maker.git
```

Note that the first repository requires that you supply the `--recurse-submodules` option to the clone command so that git will also fetch the source code for the CP/M OS.

You are now ready to proceed to compiling and installing the firmware.

Z80-Retro Firmware Installation Guide

Back

Prerequisites

In order to compile the firmware you must have completed the steps described in Z80-Retro Development Environment

The firmware executes on the Flash after a system reset or at power on assuming the reset logic is working correctly. Either way, its task is to copy the first 16KB from partition 1 of the SD Card into at 0xC000 then jump to 0xC000 and begin execution from there. The low memory bank is set to bank 14 at this time.

Compiling Firmware

The firmware is in the 2063-Z80-cpm repository.

```
$ cd 2063-Z80-cpm
$ git checkout main
$ cd boot
$ make world

... build output ...

$ ls -gG
total 108
-rw-r--r-- 1   407 Apr 18 11:25 Makefile
-rw-r--r-- 1   305 Apr 18 11:25 README.md
-rw-r--r-- 1 11856 Apr 18 11:25 firmware.asm
-rw-r--r-- 1  2471 Apr 18 11:28 firmware.bin
-rw-r--r-- 1 80818 Apr 18 11:28 firmware.lst
-rw-r--r-- 1  1498 Apr 18 11:28 firmware.sym
```

The `firmware.bin` file is ready in the `boot/` directory to be burned on to your flash chip now.

Installing Firmware

Installing the firmware requires that you have a some means by which to program the SST39SF010 flash chip. You can follow the guides to build the Raspberry Pi programmer board and hook that up to the Z80-Retro or you can program the chip using one of the commercially available chip programmers.

Typically, you won't need to program the flash that often unless you are intending to develop and test custom firmware. The process for that is beyond the scope of this manual.

Using the Raspberry Pi Programmer

The main benefit of using this programmer board is that you can program your flash in-system. That means that you don't need to pull the flash chip from the Z80-Retro to program it. It takes control of the Z80 address and data buses while it programs the flash.

The process for this is described well in these videos.

- *Video Link: Z80 Retro #9 - FLASH Programmer*
- *Video Link: Z80 Retro #10 - FLASH Programmer Software*

Using an XGPro TL866ii Plus

This method requires that you carefully pull the flash chip out of the Z80-Retro. Be careful as it's very easy to bend the IC Pins when pulling chips out of their sockets.

The manufacture supplied software for the XGPro runs on windows only. There is an open source project on GitLab that you can clone, build and use to program your chips. Follow the usage instructions in that repository.

Z80-Retro Software Set-up Guide

Back

The Retro file system provided by the example file system GitHub repository contains some useful utilities needed to get going.

There are many other CP/M software resources on the internet that are usually compatible with the Z80-Retro!

This guide explains how to set up the SD Card with CP/M.

Compile CP/M

The first step is to compile CP/M and create a binary image of the CP/M drive that you can copy to your SD Card.

```
$ cd 2063-Z80-cpm
$ git checkout main
$ make world

... build output ...

$ ls -gG boot/firmware.bin
-rw-rw-r-- 1 2472 May 24 12:34 boot/firmware.bin

$ ls -gG retro/retro.bin
-rw-rw-r-- 1 15196 May 24 12:34 retro/retro.bin

$ ls -gG filesystem/retro.img
-rw-rw-r-- 1 407552 May 24 13:04 filesystem/retro.img
```

After preparing the SD Card the `retro.bin` image can be copied to it.

Preparing the SD Card

The process for this step is described in the README-SD.md.

WARNING: This process can overwrite your entire host PC volume. Proceed carefully.

In summary the steps are as follows:

- Find the SD Card device name

  There are a few different ways to do this. The most common method on a linux system is to run the `dmesg` command which shows kernel events. When you insert a card into the SD Card slot on your system, you will find references to it in the output of a dmesg command call.

```
$ dmesg | tail -n 10
[1544787.364682] sd 6:0:0:1: [sdd] 30916608 512-byte logical blocks:
    (15.8 GB/14.7 GiB)
[1544787.365736] sd 6:0:0:1: [sdd] Write Protect is off
[1544787.365738] sd 6:0:0:1: [sdd] Mode Sense: 23 00 00 00
[1544787.366475] sd 6:0:0:0: [sdc] Attached SCSI removable disk
[1544787.367845] sd 6:0:0:1: [sdd] Write cache: disabled, read cache:
    disabled, doesn't support DPO or FUA
[1544787.385887]  sdd: sdd1
[1544787.398816] sd 6:0:0:2: [sde] Attached SCSI removable disk
[1544787.412898] sd 6:0:0:1: [sdd] Attached SCSI removable disk
[1545018.496679]  sdd: sdd1
[1545023.074094]  sdd: sdd1
```

Here, on my system, I can see that a 16GB disk was mounted on sdd. You can safely verify by running `fdisk --list /dev/sdd` where `sdd` is the device name you found in the output of `dmesg`.

```
$ sudo fdisk --list /dev/sdd
Disk /dev/sdd: 14.76 GiB, 15829303296 bytes, 30916608 sectors
Disk model: Micro SD
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00000000

Device     Boot Start      End  Sectors  Size Id Type
/dev/sdd1        8192 30916607 30908416 14.8G  c W95 FAT32 (LBA)
```

Now that you are sure that /dev/sdd is the device you want to partition, you can proceed. Be careful to always use this device name on the same system. If you choose to run this process on a different system, make sure that you first verify the device name with `dmesg` and `fdisk --list`

- Wipe out the master boot record

```
sudo dd if=/dev/zero of=/dev/sdd bs=512 count=10
```

- Create a single partition on the SD Card.

  (Copied from the README-SD.md on the repo - but ensureing that the drive is correct for this author's system.)

```
sudo parted /dev/sdd
(parted) mklabel msdos
(parted) mkpart primary 1 135
(parted) print
Model: Generic MassStorageClass (scsi)
Disk /dev/sdd: 15.6GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End    Size   Type     File system  Flags
1      1049kB  135MB  134MB  primary               lba

(parted) q
```

Copy CP/M to the SD Card

This step describes how to copy the base OS onto the SD Card.

- ☐ TODO: Add section with instructions building multiple CP/M drives using the z80-disk-maker repository.

The `make world` recipe compiles 3 main features:

- boot/firmware.bin - Firmware for the FLASH ROM.
- retro/retro.bin - BIOS and Base OS
- filesystem/retro.img - BIOS and Base OS plus some other utilities. - Drive A.

You could go ahead and copy just the BIOS + CP/M onto the SD Card, but without some of the extra utilities it could be difficult to get started. Our recommendation is to copy the `retro.img` which contains some of the example programs that you have seen John talk about in his videos.

Using the included Makefile to copy drive.img to your SD Card

The `filesystem/Makefile` does include a burn recipe that is designed to be run on a Raspberry Pi. Specifically, John's Raspberry Pi. This guide explains how to work with your SD Card on your system using the Linux commands directly.

If you wish to use the Makefile and the `make burn` recipe, you will need to alter the `burn` recipe to include the correct hostname and disk device name for your system.

Using native Linux commands to copy the drive.img to your SD Card

Copy `retro.img` to your partitioned SD Card using the dd command.

```
1 $ cd 2063-Z80-cpm
2 $ sudo dd if=filesystem/retro.img of=/dev/sdd1 bs=512 conv=sync
```

Now eject your SD Card and try it out on the Retro. If everything works, you should see something like this on your terminal:

```
Z80 Retro Board 2063.3
      git: v20230312.1-16-gef7b3f7 2023-05-04 21:48:53 -0500
    build: 2023-05-13 17:21:25+12:00

Booting SD card partition 1

Partition Table:
C1BE: 00 04 01 04 83 05 82 06 00 08 00 00 00 04 00
C1CE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1DE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1EE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Partition 1 starting block number: 00000800
Partition 1 number of blocks:      00040000

Loading 0x20 512-byte blocks into 0xC000 - 0xFFFF
................................

Z80 Retro BIOS Copyright (C) 2021 John Winans
CP/M 2.2 Copyright (C) 1979 Digital Research
  git: v20230317.1-0-g8bcf644 2023-03-17 22:20:02 -0500
build: 2023-05-24 13:04:35+12:00

NOTICE: rw_dmcache library installed.

a>
```

Transfering Files to CP/M

Back

There are a few different ways one can transfer files to CP/M over the serial connection.

- Transfer them using `pip` over the existing serial connection
- Transfer them using `xmodem` over the existing serial connection

The following sections show how to compile XMODEM, transfer the binary in intel hex format and then load it as a com file inside CP/M. We then cover how to use xmodem to transfer regular binary files from then on.

Compile XMODEM

```
$ sudo apt-get install srecord

$ git clone https://github.com/Z80-Retro/xmodem80

$ cd xmodem80

$ make
```

Check that your XR.hex file looks like this.

```
$ cat XR.hex
:200100003A0200329E0232A10232A402ED737A0331040521A502CD5F023A5D00FE20CAF99F
:2001200001115C003E00327C000E16CD05003CCAE1013E01327E033EFE327F033E14327CA5
:20014000033E05CD2A02D25801217C0335CA02020E15CDA202C34101FE04CAC701FE18CA85
:200160000802FE01C241012180037723E50683C53E01CD2A02C1E17723E510F3E12183031D
:2001800006803E00862310FCAEC202023A7E034F3A8103B9C202023A7F034F3A8203B9C2E6
:2001A00002021183030E1ACD0500115C000E15CD0500FE00C2F301217E0334217F03350ED8
:2001C00006CDA202C33C01CD0E020E06CDA202212703CD5F02CD7702212F03CD5F02C3251E
:2001E00002214803CD5F02CD7702217703CD5F02C3250221DA02C31702216403CD5F02C318
:20020000250221F502C31702210C03C31702115C000E10CD0500C9CD5F02CD0E02115C0019
:200220000E13CD0500ED7B7A03C947C506FFC506FFC5CD9C02FE00C257022A00002A0000A5
:200240002A00002A00002A00002A0000C110E2C110DCC110D637C9CD9F02C1C1C1B7C97E40
:20026000B7C84FCDA20223C35F02C57E4FFE20C4A202C12310F4C93A5C00B7CA8902C64087
:200280004FCDA2020E3ACDA202215D000608CD6A020E2ECDA2020603CD6A02C9C306FFC3DD
:2002A00009FFC30CFF43502F4D205852202D20586D6F64656D2072656365697665207630EF
:2002C0002E32202F20536D616C6C526F6F6D4C6162732032303137 0D0A000D0A4661696CA3
:2002E000656420777269746 96E6720746F206469736B0D0A000D0A5472616E736D73736951
:200300006F6E206661696C65640D0A000D0A5472616E736D697373696F6E2063616E636529
:200320006C6C65640D0A000D0A46696C652000207265636569766564207375636365737 3CE
:2003400066756C6C790D0A004661696C656420637265617469 6E672066696C65206E616D8C
:20036000656420004669 6C656E616D652065787065636564 0D0A000D0A0000000000000047
:0405000000000000F7
:00000001FF
```

Copy XMODEM to the Z80-Retro! Using PIP

You can use `pip` to copy files on CP/M between user areas and drives. You can also use it to transfer files from the host PC over the serial connection.

This example shows how to copy the xmodem binary in intel hex format from the host PC to the A: drive.

It assumes you have a copy of the xmodem utility in intel hex format already.

First you will want to set-up your terminal to use a 1msec delay between chars. If you past at full speed, the terminal will not keep up and you will loose data.

Set the Transmit delay to 1 msec/char and click New setting. You might also want to set the msec/line setting to 1 in case you encounter errors with the process below.

Minicom also as a way to do this. Read the manual. These instructions are for Terra Term on Windows.

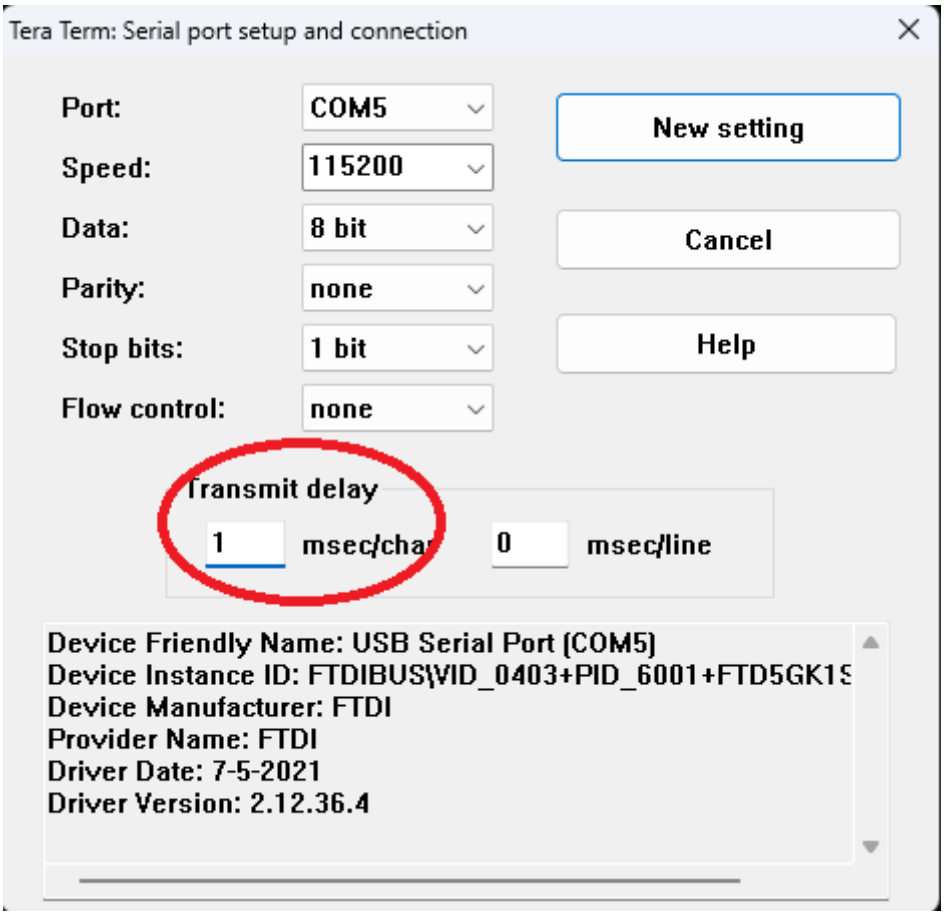

Figure 11: Terraterm Settings

When you are ready type: `pip xr.hex=con:` and then paste the hex data.

When the paste is finished, go ahead and press `CTRL+Z` to indicate to pip that there is no more data to send.

```
a>pip xr.hex=con:
<paste hex data>
:200100003A0200329E0232A10232A402ED737A0331040521A502CD5F023A5D00FE20CAF99F
:2001200001115C003E00327C000E16CD05003CCAE1013E01327E033EFE327F033E14327CA5
:20014000033E05CD2A02D25801217C0335CA02020E15CDA202C34101FE04CAC701FE18CA85
:200160000802FE01C241012180037723E50683C53E01CD2A02C1E17723E510F3E12183031D
:2001800006803E00862310FCAEC202023A7E034F3A8103B9C202023A7F034F3A8203B9C2E6
:2001A00002021183030E1ACD0500115C000E15CD0500FE00C2F301217E0334217F03350ED8
:2001C00006CDA202C33C01CD0E020E06CDA202212703CD5F02CD7702212F03CD5F02C3251E
:2001E00002214803CD5F02CD7702217703CD5F02C3250221DA02C31702216403CD5F02C318
:20020000250221F502C31702210C03C31702115C000E10CD0500C9CD5F02CD0E02115C0019
:200220000E13CD0500ED7B7A03C947C506FFC506FFC5CD9C02FE00C257022A00002A0000A5
:200240002A00002A00002A00002A0000C110E2C110DCC110D637C9CD9F02C1C1C1B7C97E40
:20026000B7C84FCDA20223C35F02C57E4FFE20C4A202C12310F4C93A5C00B7CA8902C64087
:200280004FCDA2020E3ACDA202215D000608CD6A020E2ECDA2020603CD6A02C9C306FFC3DD
:2002A00009FFC30CFF43502F4D205852202D20586D6F64656D207265636565697665207630EF
:2002C0002E32202F20536D616C6C526F6F6D4C61627320323031370D0A000D0A4661696CA3
:2002E000656420777269746974696E6720746F206469736B0D0A000D0A5472616E736D73736951
:200300006F6E206661696C65640D0A000D0A5472616E736D697373696F6E2063616E636529
:200320006C6C65640D0A000D0A46696C652000207265636569766564207375636365737CE
:2003400066756C6C790D0A004661696C656420637265617469E672066696C65206E616D8C
:20036000656420004669 6C656E616D652065787065636564D0A000D0A0000000000000047
:04050000000000F7<CTRL+Z>
```

Now you have a new `xr.hex` file on your disk.

You can validate it's correct by typing `type xr.hex`.

Next step is to convert the hex data into a `COM` file using the `LOAD` utiltity.

```
a>load xr.hex


FIRST ADDRESS 0100
LAST  ADDRESS 0503
BYTES READ    0284
RECORDS WRITTEN 09

a>stat xr.com


 Recs  Bytes  Ext Acc
    9     2k    1 R/W A:XR.COM
Bytes Remaining On A: 7608k
```

If you want to you can delete the original hex file.

```
a>era xr.hex
```

Use XMODEM to Transfer a File

NOTE: I don't have minicom working to test this using minicom.

- ☐ TODO: Convince someone else to write the minicom process.

The `xr.com` utility will let you transfer any files at a faster rate across the serial connection. This example shows how to load the `xs.com` binary file that was also compiled in the xmodem repository.

Start with:

```
a> xr xs.com
```

The destination file name in this case will be xs.com.

Then use your serial tool to initiate an xmodem send.

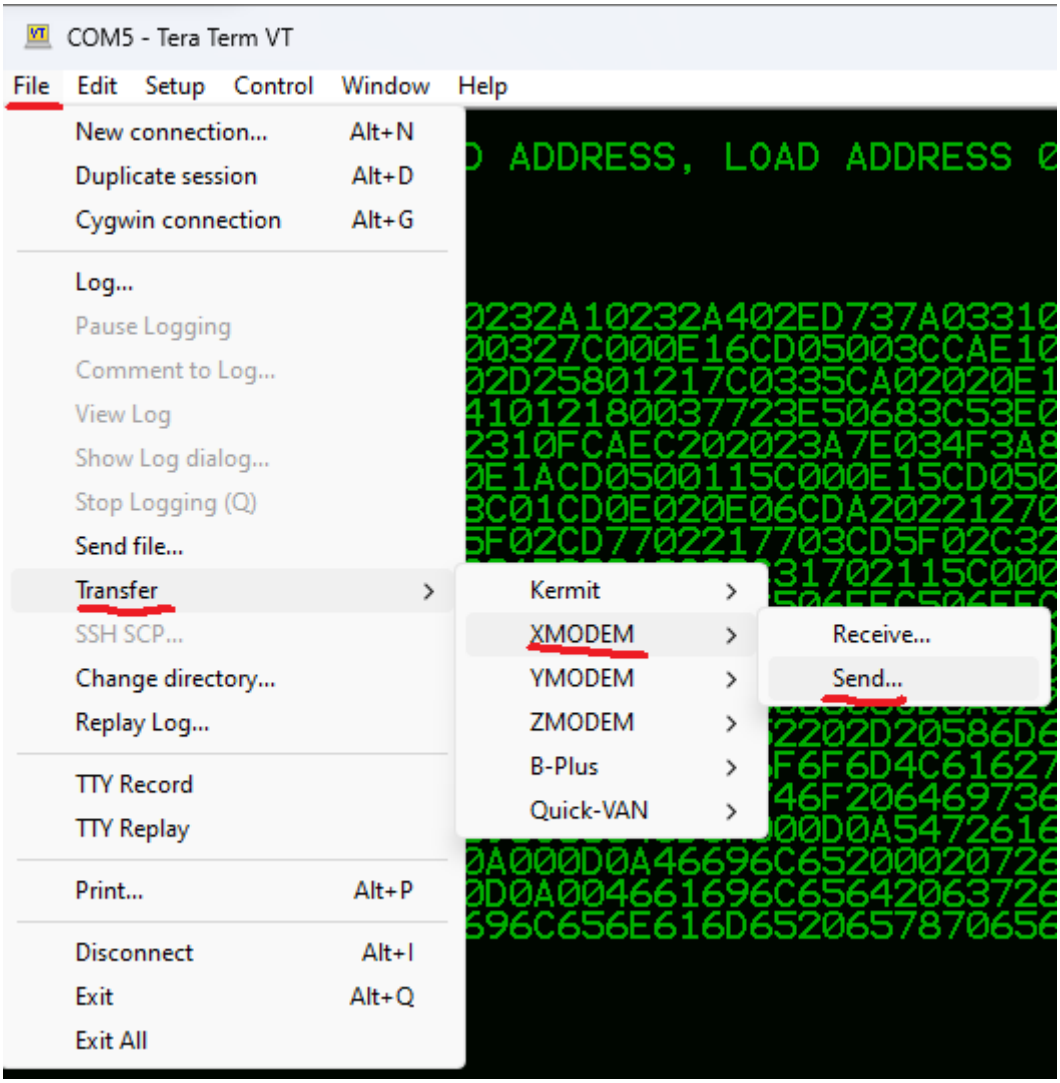

Figure 12: initiate transfer

Select the file you want to send.

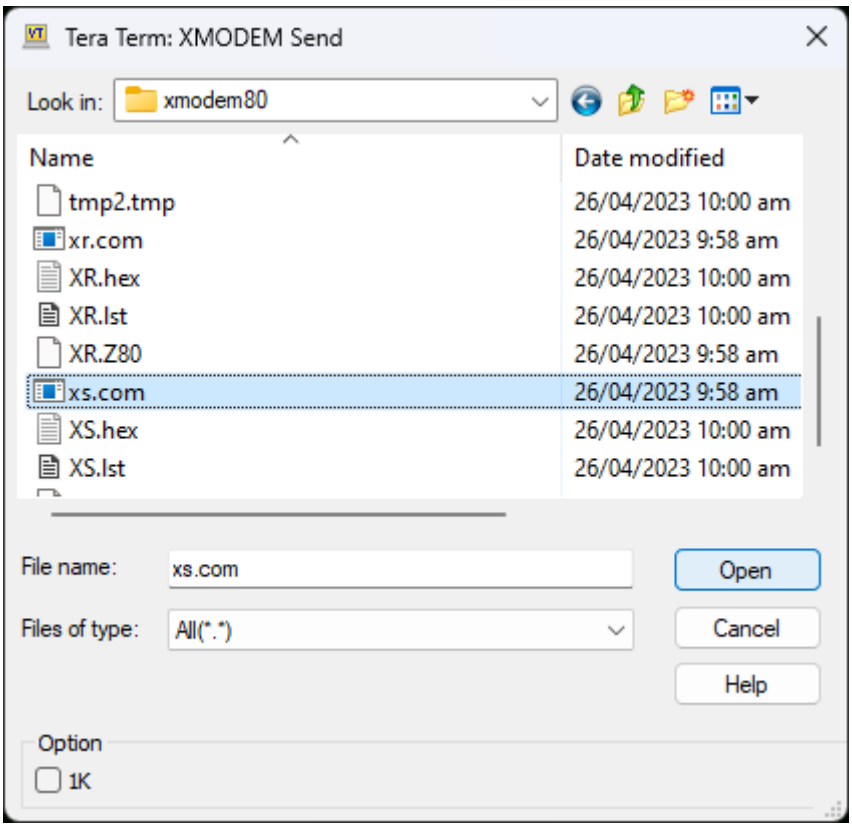

Figure 13: select file

Wait for the file to complete sending.

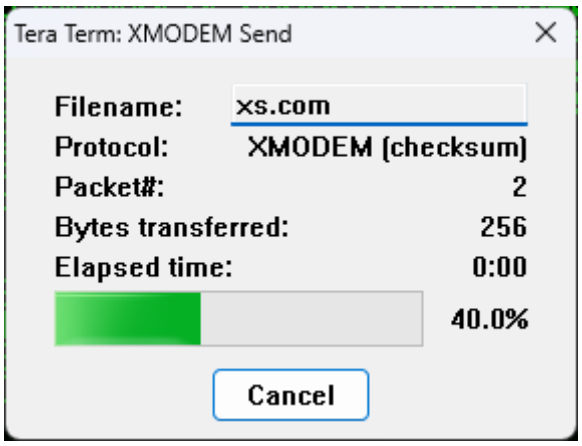

Figure 14: sending dialogue

Here is how the whole process looks inside CP/M.

```
a>xr xs.com

CP/M XR - Xmodem receive v0.2 / SmallRoomLabs 2017
<initiate transfer as shown in the pictures>
File XS.COM received successfully

a>stat xs.com


 Recs  Bytes  Ext Acc
    5     2k    1 R/W A:XS.COM
Bytes Remaining On A: 7608k
```

The `xs.com` utiltiy is for receiving files from CP/M to the host computer.

It's almost the same process as for sending files to CP/M from the host computer. Just type `xs <filename to send>` and then initiate an XMODEM receive in the terminal program.

```
a>xs xr.com
<initiate XMODEM receive, select path and name for file on host>
CP/M XS - Xmodem Send v0.1 / SmallRoomLabs 2017

File XR.COM sent successfully
```

The file will exist at the file path and name.

Z80-Retro IO Ports

Back

This table provides a list of the assigned IO ports in the Z80 Retro and some of the additional daughter boards including the VDP.

Information checked against schematics and in the CPM IO library and example VDP Code.

- 2063-z80.pdf
- 2063-z80-cpm IO Lib
- FLEADiP

Port Range	Programmed Port	Board	Description
00-0F	00	2063-Z80	GP IO In
10-1F	10	2063-Z80	GP IO Out
20-2F	20	2063-Z80	Printer Data Out
30-3F	30	2063-Z80	SIO Port A - Data
30-3F	31	2063-Z80	SIO Port B - Data
30-3F	32	2063-Z80	SIO Port A - Control
30-3F	33	2063-Z80	SIO Port B - Control
40-4F	40	2063-Z80	CTC Port 0
40-4F	41	2063-Z80	CTC Port 1
40-4F	42	2063-Z80	CTC Port 2
40-4F	43	2063-Z80	CTC Port 3
50-5F	50	FLEADiP	FPGA Address Register
50-5F	51	FLEADiP	FPGA Data Register
50-5F	58-5F	Z80 Retro TL16C550C Card	16550 Registers
60-6F			
70-7F	70	2063-Z80	Flash Disable
80-8F	80	2068-Z80-TMS9118 / FLEADiP	TMS9118 VRAM
80-8F	81	2068-Z80-TMS9118 / FLEADip	TMS9118 REGISTER
90-9F		2068-Z80-TMS9118	
A0-AF	A4	FLEADiP	ESP Programmer
A0-AF	A8	2068-Z80-TMS9118 / FLEADiP	Joystick 0 (J3)
A0-AF	A9	2068-Z80-TMS9118 / FLEADiP	Joystick 1 (J4)
B0-BF			
C0-CF			

Port Range	Programmed Port	Board	Description
D0-DF			
E0-EF			
F0-FF			

Z80-Retro VDP Build Guide

The VDP Daughter Board is designed to be mounted on to the 40pin header on the main board. You can find the hardware build repository here: 2068-Z80-TMS9118

- Videos:
  - *Video Link: NTSC Video Basics*
  - *Video Link: Z80 Retro #67 - Composite to HDMI Converter/Adapter*
  - REV 0 - DO NOT BUILD:
    * *Video Link: Z80 Retro #44 - Contemplating the Addition of a TMS9118 Video Display Processor*
    * *Video Link: Z80 Retro #45 - TMS9118 VDP Breadboard Testing*
    * *Video Link: Z80 Retro #47 - TMS9118 VDP Breadboard Testing The Z80 Retro! Interface*
    * *Video Link: Z80 Retro #49 - TMS9118 VDP First Draft Schematic & PCB for the Z80 Retro!*
    * *Video Link: Z80 Retro #48 - TMS9118 VDP Rev 0 PCB Build Spoiler: DO NOT BUILD!*
    * *Video Link: Z80 Retro #50 - TMS9118 VDP Rev 0 PCB Test pt. 1*
    * *Video Link: Z80 Retro #51 - TMS9118 VDP Rev 0 PCB Test pt. 2*
    * *Video Link: Z80 Retro TMS9118 VDP Breaking News!*
    * *Video Link: Z80 Retro #52 - TMS9118 VDP Rev 0 PCB Test pt. 3 (Joystick)*
    * *Video Link: Z80 Retro #53 - TMS9118 VDP Rev 0 PCB Test pt. 4 (IRQs)*
    * *Video Link: Z80 Retro #54 - TMS9118 VDP Rev 0 PCB Test pt. 4a (IRQs and DDT)*
  - REV 1:
    * *Video Link: Z80 Retro #55 - TMS9118 VDP Rev 1 PCB Test*
    * *Video Link: Z80 Retro #56 - TMS9118 VDP Graphics Mode 1*
    * *Video Link: Z80 Retro #57 - TMS9118 VDP Sprites*
    * *Video Link: Z80 Retro #58 - TMS9118 VDP Frame Flag Race Condition?*
    * *Video Link: Z80 Retro #71 - VDP Board Issues Posted to Github*
  - REV 2:
    * *Video Link: Z80 Retro #72 - VDP Board Rev 2 Upgrade* comming soon to a youtube near you.

The same guidelines as noted in: Z80-Retro Hardware Build Guide apply here.

The README in the hardware project provides a detailed BOM and information on where to source the TMS9118A processor from. Note: You can *NOT* use a TMS9918A in this build as it requires 8x4k RAM chips. The TMS9118A only requires 2x16k ram chips which are easier to find and allow for a smaller board footprint.

Mounting the board

When you go to mount the board on on to the main board, pay careful attention to the clearance below. Make sure that the crystal oscillators do not touch any of the solder joints on the VDP board.

The mounting holes on the right hand side of the VDP board are located such that stand-offs can be installed to ensure a robust clearance below. Alternatively, you can insert a layer of cardboard between the boards.

Should you wish to deploy the Raspberry PI Programmer Board on top of the VDP, please check your clearances between the programmer and the components on VDP. In particular, the electrolytic capacitor at C7 could be too high.

This is discussed in the “Z80 Retro #71 - VDP Board Issues Posted to Github” video linked above.

In any event, due to the thermodynamics of these TMS chips discussed in the next paragraph, you should not leave the programmer board permanently connected above the VDP.

Heat

Up until now, everything you know and have learned about CMOS circuits will be that they should be cool to the touch when in operation. This is not the case with the TMS9118A or the two ’1416 64kbit RAM chips. They will get quite warm. Not so hot as to burn you, but warm to the tough. Keep the boards well ventilated and you will be fine. If you want, to you can add some heat sinks to these 3 chips. If you intend to install the Z80-Retro! into an enclosure, it is recommended to install heat sinks and even some active airflow.

As Built Image

NOTE: *I do not have J5 Populated.*

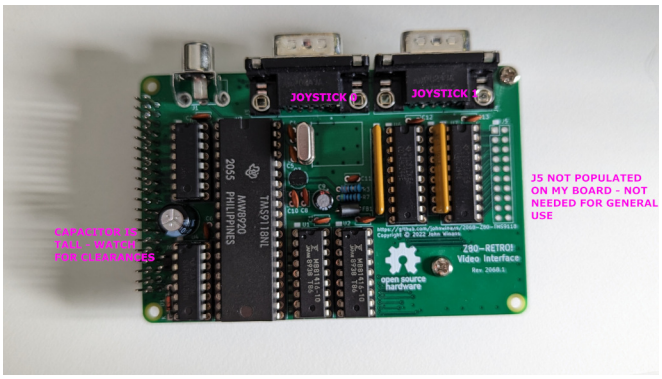

Figure 15: As built photo of VDP Board

Figure 16: Detail of J6 not shunted and tall capacitor

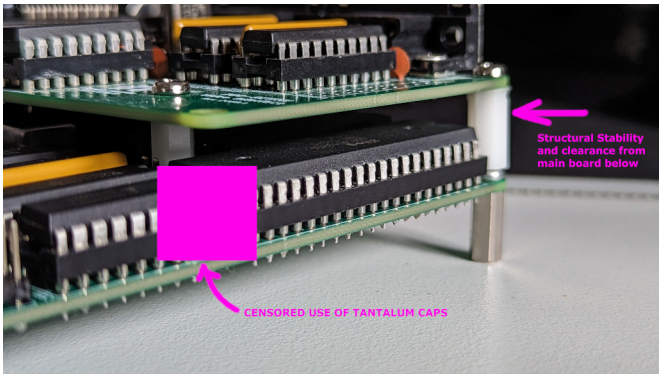

Figure 17: Showing the plastic sand-offs

The Rev 2 Bodge

The main reason for developing and including the TMS VDP Board is to play games on the Z80-Retro! Games of any era rely heavily on accurate timing for both frame rendering and smooth game play.

The typical way that this is achieved on the TMS9118A, is through observing the VSYNC Status of the device. The TMS9118A generates a low going interrupt signal on pin 16 every 60th (1/60) of a second. Thus if the software running on the retro is able to keep up, one could conceivably have a game that renders at 60FPS! Not bad for devices made over 40 years ago!

The Z80-Retro! does not enable interrupts in any way in the default CP/M BIOS.

It is perfectly acceptable to develop for interrupt usage, so long as you pick Interrupt Mode 1 on the Z80 as this is the only interrupt mode that will work with the TMS9118A.

Another way to “notice” when the TMS9118A is asserting the interrupt is to poll the Status Register on the TMS until it returns 1 on the most significant bit.

In C this looks something like this.

```
1  while (IO_VDPLATCH == 0) { ; }
```

Unfortunately, testing and measuring has shown this method to often result in skipped frames which results in very poor game performance. This is probably because the act of reading the register also clears the interrupt status. Thus the race condition discussed in some detail here: - *Video Link: Z80 Retro #58 - TMS9118 VDP Frame Flag Race Condition?*

As a workaround to this and still not using interrupts, we can apply a small bodge to the REV 1 VDP Board, which allows for programs to monitor the status of the interrupt PIN on the TMS9118A without reading the status flag. Turns out the Joystick Buffers have some free bits available that are not used by the joy- stick ports. We can assign one of the free pins on Joystick 0 (IO PORT A8) to this purpose.

Now with the bodge applied, the test for vertical sync interrupt looks like this:

```
1  void vdp_waitVDPReadyInt() {
2    while ((IO_JOY0 & 0x02) != 0)
3      ;
4    vdpStatusRegVal = IO_VDPLATCH;
5  }
```

This works because the Joystick Buffers have all of their inputs pulled up by 10k resistors. When the TMS pulls the line low, the result of the bit mask will be 0 and we can *know* that a vertical sync event has occurred and use this data to time everything in our games.

Here is an image of the bodge applied.

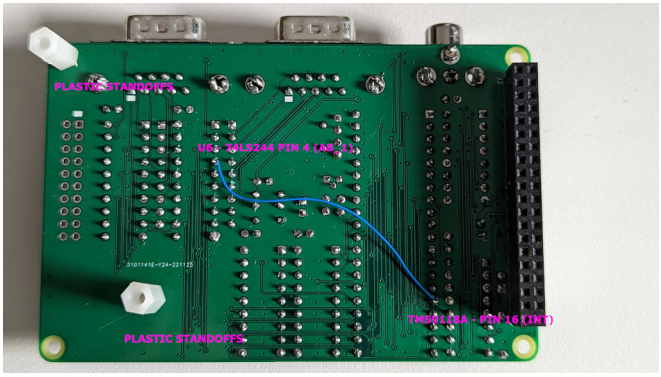

Figure 18: REV 2 Bodge

TODO: [ ] Add link to video that describes the bodge and demo's the results of applying it to games.

The REV 2 Board currently in development provides this link in the copper traces as well as offering a new Jumper (J7) to optionally enable or disable it.

Back

Using the Etched Pixels Emulator Kit

Back

The Etched Pixels Emulator kit supports the “Z80-Retro!” along with many other homebrew projects.

Clone and Build

Clone the Etched Pixels Emulator Kit github repository here from here: https://github.com/EtchedPixels/EmulatorKit

```
$ cd EmulatorKit
$ make
< compilation takes a while>
$ cp 2063 ../2063-Z80-cpm
```

Compilation takes a while as it compiles all the emulators for the different supported projects.

Now that you have the emulator binary you should be ready to go.

Preparing the firmware and the sdcard image

You will need to prepare the firmware image and the SD Card image for use with the emulator. As we don’t have a real formatted SD Card to work with we need to create an SDcard.img file that will have the correctly formatted partitions on it.

Create the ROM Image

The emulator expects the firmware image to be exactly 16KB in size. Use the `truncate` command in Linux to do that.

```
$ cd 2063-Z80-cpm/boot
$ make
$ truncate -s 16K firmware.bin
$ hexdump -C firmware.bin| tail -n 3
000009b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
    |................|
*
00004000
```

Validate that the last line is 00004000 (0x4000 in hex) or 16K.

Create the SD Card Image

!!! WARNING !!! This process can damage your host file system if you get it wrong. Make sure you only format the loop device after setting it up.

This step is a bit more complicated and these steps are definitely ONLY going to work on a Linux host.

```
$ cd 2063-Z80-cpm/filesystem

$ make

$ dd if=/dev/zero of=SDcard.img bs=1024 count=512000

$ sudo losetup -Pf SDcard.img

$ (
echo n  # Add a new partition
echo p  # Primary partition
echo 1  # Partition number
echo    # First sector (Accept default)
echo +128M # Last sector (Accept default: varies)
echo n  # Add a new partition
echo p  # Primary partition
echo 2  # Partition number
echo    # First sector (Accept default)
echo    # Last sector (Accept default: varies)
echo t  # Partition Type
echo 1  # First partition
echo db # Tpye = CP/M
echo t  # Partition Type
echo 2  # Second partition
echo 06 # Fat 16
echo w  # Write changes
) | sudo fdisk /dev/loop0
```

Whew! Now that you have an image file with partitions on it, you can copy the `retro.img` CP/M file system on to it.

```
$ sudo mkfs.msdos /dev/loop0p1
$ sudo mkfs.msdos /dev/loop0p2
$ sudo dd if=retro.img of=/dev/loop0p1 bs=512
```

Finally unmount the loopback disk.

```
$ sudo losetup -d /dev/loop0
```

You should now have a 500MB SDcard.img file.

Running the Emulator

To distinguish this “Z80-Retro!” project from a different Z80-Retro project, the author has named the emulator `2063` after the board number.

```
$ cd d 2063-Z80-cpm
$ ./2063 -r boot/firmware.bin -S filesystem/SDcard.img

Z80 Retro Board 2063.3
      git: v20230317.1-0-g8bcf644 2023-03-17 22:20:02 -0500
    build: 2023-04-23 18:03:16+12:00

Booting SD card partition 1

Partition Table:
C1BE: 00 20 21 00 7F 71 21 10 00 08 00 00 00 04 00
C1CE: 00 71 22 10 06 BC 3D 3F 00 08 04 00 00 98 0B 00
C1DE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1EE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Partition 1 starting block number: 00000800
Partition 1 number of blocks:      00040000

Loading 0x20 512-byte blocks into 0xC000 - 0xFFFF
................................

Z80 Retro BIOS Copyright (C) 2021 John Winans
CP/M 2.2 Copyright (C) 1979 Digital Research
  git: v20230317.1-0-g8bcf644 2023-03-17 22:20:02 -0500
build: 2023-04-22 17:56:49+12:00

NOTICE: rw_dmcache library installed.

a>
```

Exit the emulator by typing CTRL+\

Z80-Retro Org and Git

Back

The “Z80-Retro!” project is delivered on GitHub as an open source hardware and software project. As development of the project progresses, the various repositories are updated inside GitHub. It helps to have a basic understanding of how to use Git in this context.

Z80-Retro Organisation

The Z80-Retro Org in Github contains all the repositories (including this one) that relate to building a “Z80-Retro!” of your own. In order to maintain some modicum of sanity there are multiple repositories - each dealing with a different aspect of the project. For example, there are hardware repositories that contain the schematics and PCB KiCAD files etc. and there are software repositories that contain the firmware, CP/M and other utilities.

In general you will need to clone only a few of these repositories to your host PC to be productive. The rest of this page contains some detail on how to learn more about git, as well some resources on how the “Z80-Retro!” org will expect collaboration to occur should that be what you want to do.

Using Git

The sections below cover some of the basics. For a more detailed introduction to Git, GitHub and version control systems in general, it is suggested you review these resources.

Git SCM:

- https://git-scm.com/docs/user-manual
- https://git-scm.com/docs/gittutorial

GitHub:

- https://docs.github.com/en
- https://docs.github.com/en/get-started/using-git
- https://docs.github.com/en/get-started/quickstart/hello-world

John’s Basement

- *Video Link: Git - The Basic Concepts of Repos, Cloning, Pushing, and Pulling*
- *Video Link: NIU CSCI 340, How to set up SSH & Github*

In case you are looking to collaborate you need to familiarise yourself with the pull request work flows in GitHub. See:

- https://docs.github.com/en/get-started/quickstart/github-flow
- https://docs.github.com/en/get-started/quickstart/fork-a-repo

Deep Dive

If you're looking for a deep dive into Git concepts try:

- *Video Link: Git For Ages 4 and Up*

Collaboration Using the Fork / Pull Reuqest Workflow

We use the Fork model for dealing with pull requests. The basic flow is:

1. Fork the repository into your own Github account. (you only need to do this once)
2. Clone your fork of the repository on to your workstation.
3. Create a branch.
4. Develop and test your changes. Document anything that's new. Okay to use multiple commits.
5. Raise a pull request against the upstream project.
6. Your pull request should include a written description describing what your intentions are and why you made the changes you did.
7. The owner of the upstream project will review your changes, add any comments and generally collaborate with you on your pull request. During this time, you might add further commits to your branch and push them up to your fork. GitHub will automatically reflect this changes in the pull request so the upstream maintainer can see what you've done.
8. At some point, the maintainer will merge your pull request into the main branch of the upstream repository.
9. You will need to then rebase your fork main branch to match upstream in readiness for future pull requests and to ensure your fork is aligned with upstreeam.