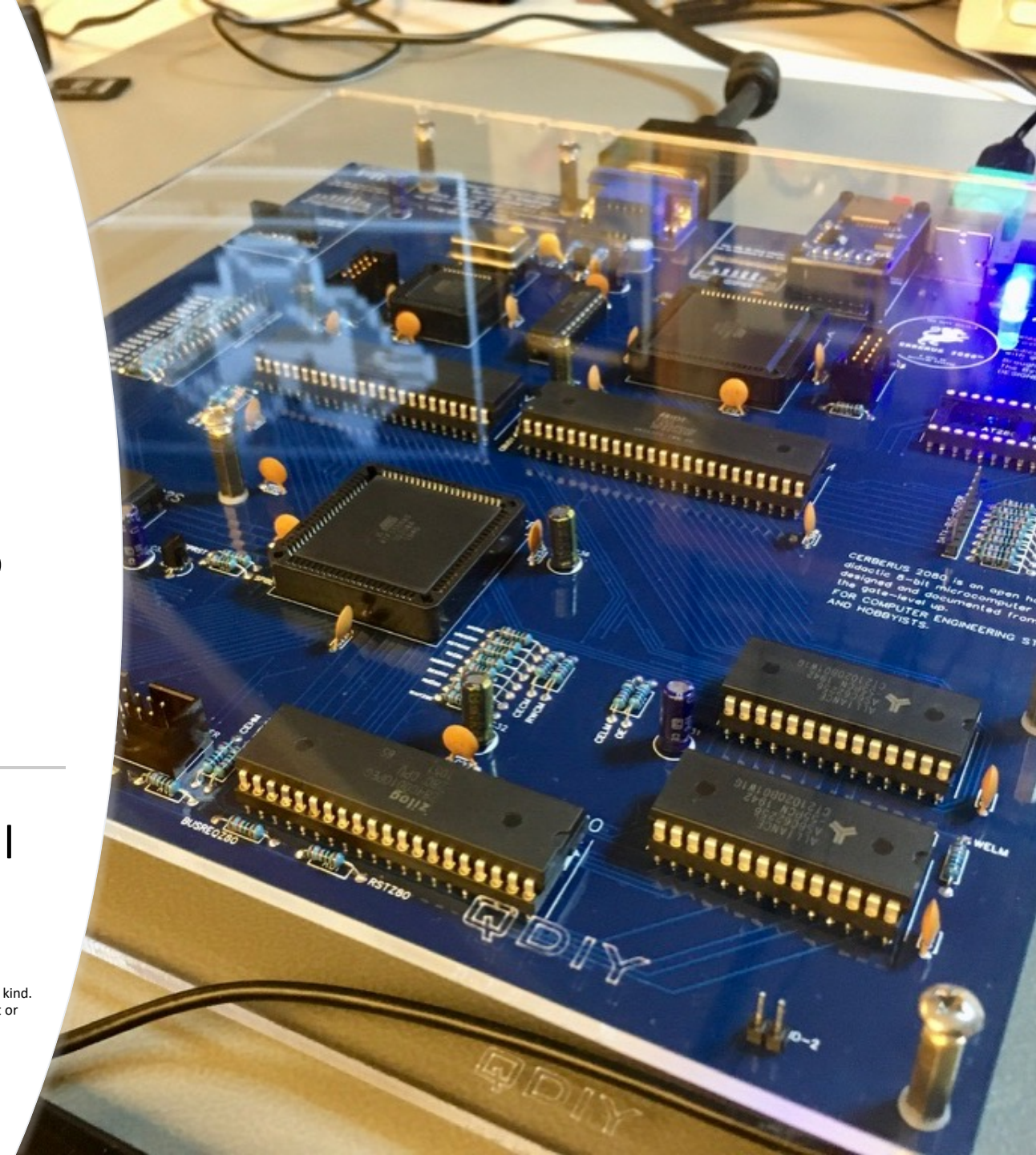
The Byte Attic's

CERBERUS 2080™

© 2021 by Bernardo Kastrup



The Byte Attic's

CERBERUS 2080™

The Complete Manual

Last updated on: 5/2/21 10:36:43 PM

Introduction

- CERBERUS 2080 is an open-source *educational platform* for students of computer engineering and advanced hobbyists
- It gives the user direct, convenient and unrestricted access to the hardware
- It aims to demystify computers by showing in detail how one is built, from the gate-level up
- CERBERUS 2080 is an innovative, *fully-functional computer*, not a toy
- Its architecture illustrates how a multi-processor system can be built
- Its design was done explicitly at the gate-level, with no high-level hardware synthesis tools
- All custom ICs are implemented by in-system programmable CPLDs
- CERBERUS 2080's architecture is clean and highly didactic, no ugly compromises having been made for cost-reduction purposes
- It can be regarded as an advanced, elegant successor to classical late-70s microcomputers such as the ZX80, TRS-80 and PET 2001

Technical specifications

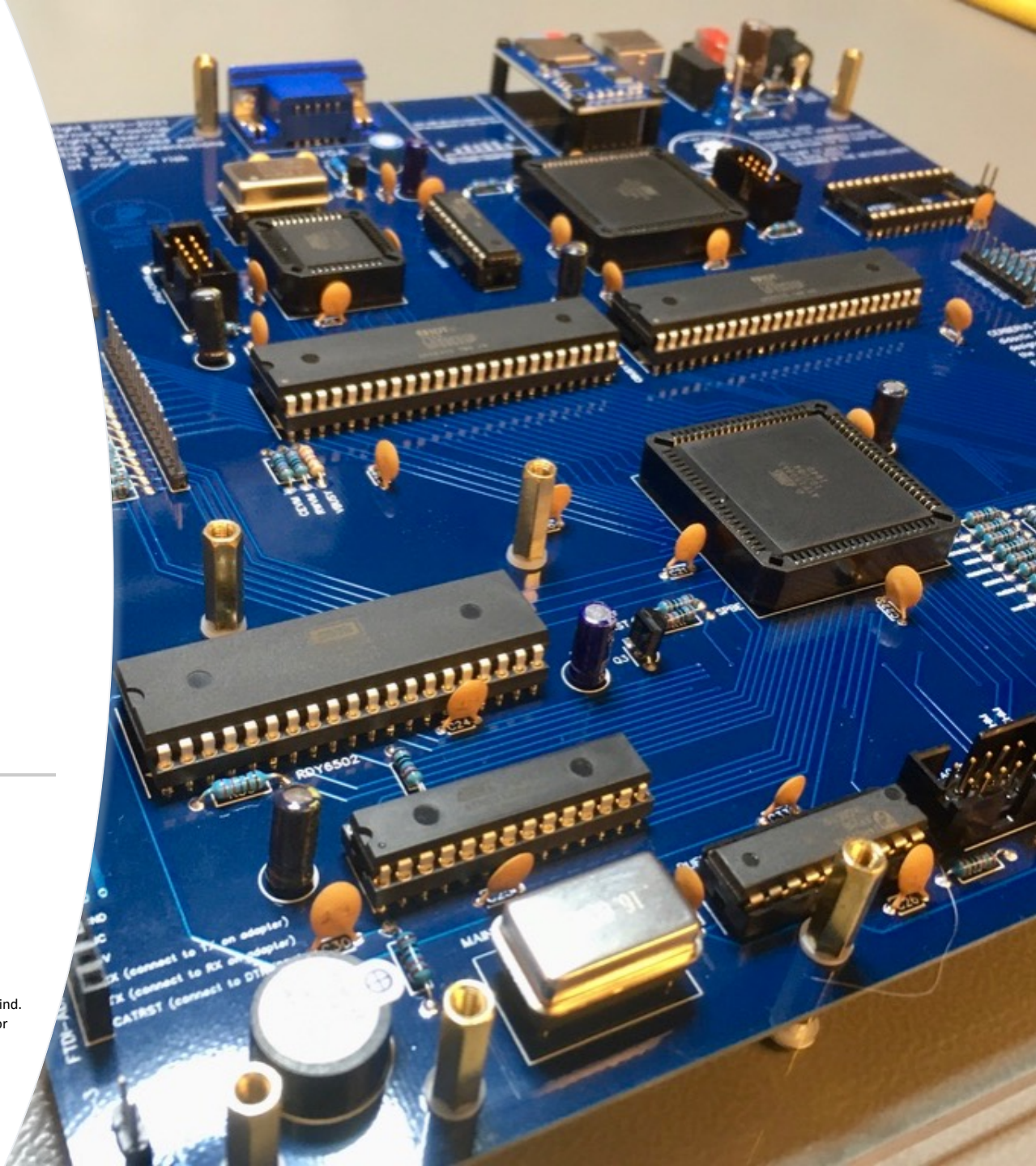
- Complete 8-bit multi-processor microcomputer
- 5V, through-hole technology
- 3 processors: **Z80** and **W65C02S** CPUs, plus **AVR RISC controller**
- CPUs run at 4 or 8 MHz (user-selectable), AVR controller at 16 MHz
- Chipset with 3 custom ICs (5V CPLDs): **SCUNK™**, **CAVIA™** and **SPACER™**
- Beeper sound (available only to the BIOS)
- Standard PS/2 keyboard
- Standard (µ)SD card storage, with file system built into the BIOS
- 64 KB of user-addressable RAM
- No ROM: BIOS is stored in AVR controller's internal Flash and uses up no address space
- Standard VGA video, character-based, 320x240 pixels (40x30 individually addressable characters), black and white
- On-the-fly user-redefinable character bitmaps

The Byte Attic's

CERBERUS 2080™

© 2021 by
Bernardo Kastrup

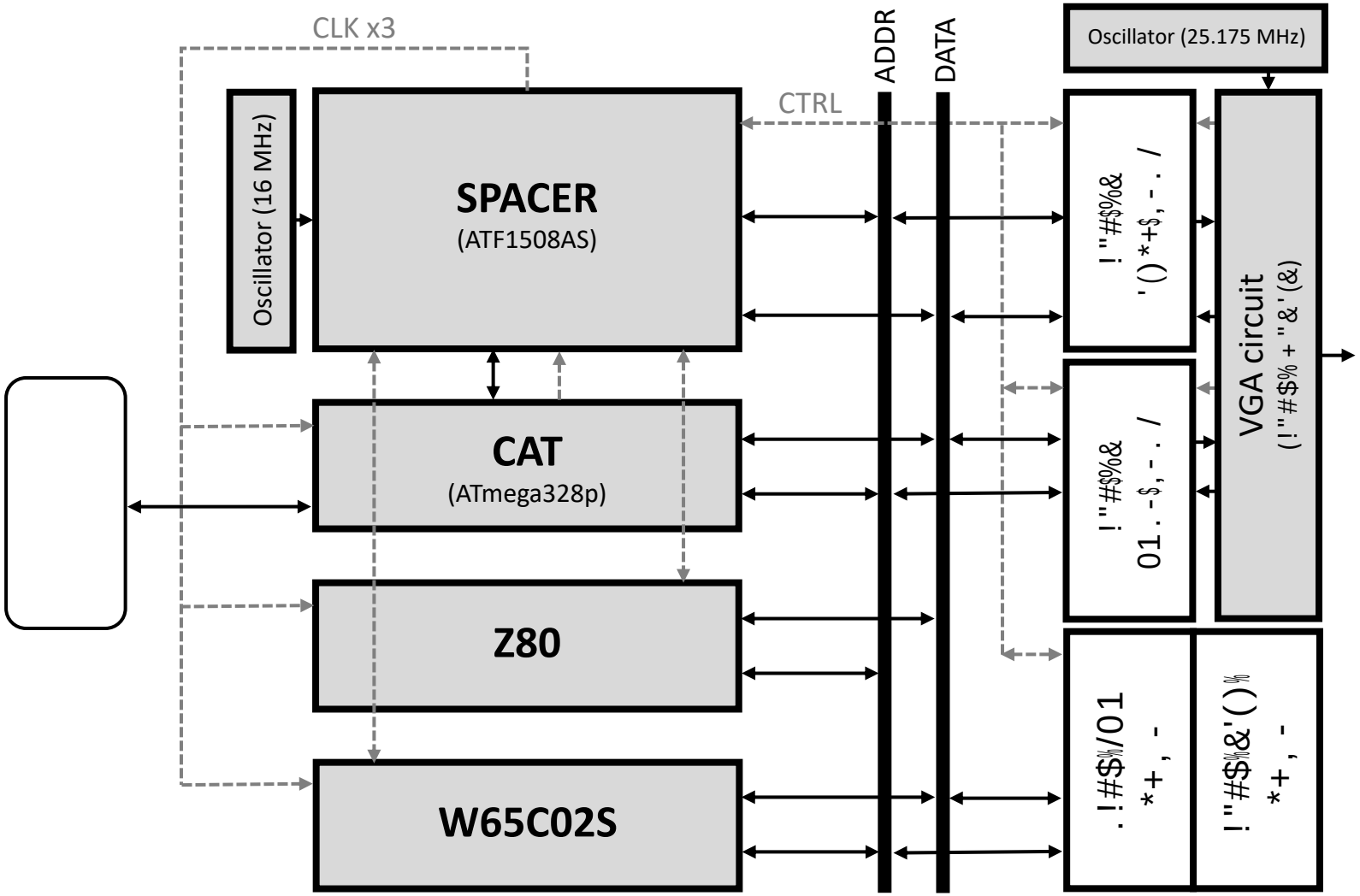
Technical Overview



System overview

- CERBERUS 2080 can be divided into two segments:
  - The *VGA circuit*, driven by a 25.175 MHz oscillator
  - The *computer proper*, driven by a 16 MHz oscillator
- The two segments are entirely asynchronous and communicate via two dual-ported memories:
  - A 2KB **video SRAM**, storing a character identifier for each screen position
  - A 2KB **character SRAM**, which holds the character definitions or bitmaps
- Two custom ICs in the VGA circuit:
  - **SCUNK** ('Scan CoUNter and clocK'), which controls all VGA timing
  - **CAVIA** ('ChAracter Video Adapter'), which continuously scans the video and character memories to generate the screen
- The computer proper consists mainly of four ICs:
  - **CAT** ('Custom ATmega328p'), the I/O processor and system master
  - **SPACER** ('Serial to PArallel ControllER'), which manages all control signals, clocks, and translates CAT's serial data into parallel words & vice-versa
  - **Z80**: one of the system's two CPUs, responsible for running applications
  - **W65C02S**: the other CPU
- As the system master, CAT runs the BIOS code (Basic Input/Output System) and delegates applications to the CPUs
- Two **32KB SRAMs** (60KB user-addressable) serve as system memory
- *No ROM*: BIOS code is stored in CAT's internal Flash memory (32 KB) and *doesn't use up system address space*

System architecture



CAT™ overview

- An ATmega328p microcontroller configured to use a strong external oscillator (16 MHz) through changes in its internal registry
- CAT is CERBERUS 2080's *system master*: it runs the BIOS code from its onboard Flash memory and controls the CPUs
- The BIOS code is written in C and compiled under the Arduino IDE
  - From the IDE's perspective, CAT is regarded as an Arduino UNO board
- Except for video output, CAT performs all I/O functions: file system operations, keyboard control and sound output
- CAT is capable of DMA (Direct Memory Access) through SPACER
  - Although CAT is a serial controller, through SPACER's internal shift registers it can access both data and address buses
- The user can only access the CPUs through CAT
  - It resets, selects, starts, halts, interrupts, passes on keyboard inputs & delegates applications to the CPUs via SPACER
- CAT determines the CPU clock frequency (4 or 8 MHz)
- Because of its serial nature, CAT is very slow compared to the CPUs, but excels in flexibility and is therefore suitable for I/O operations and global system control

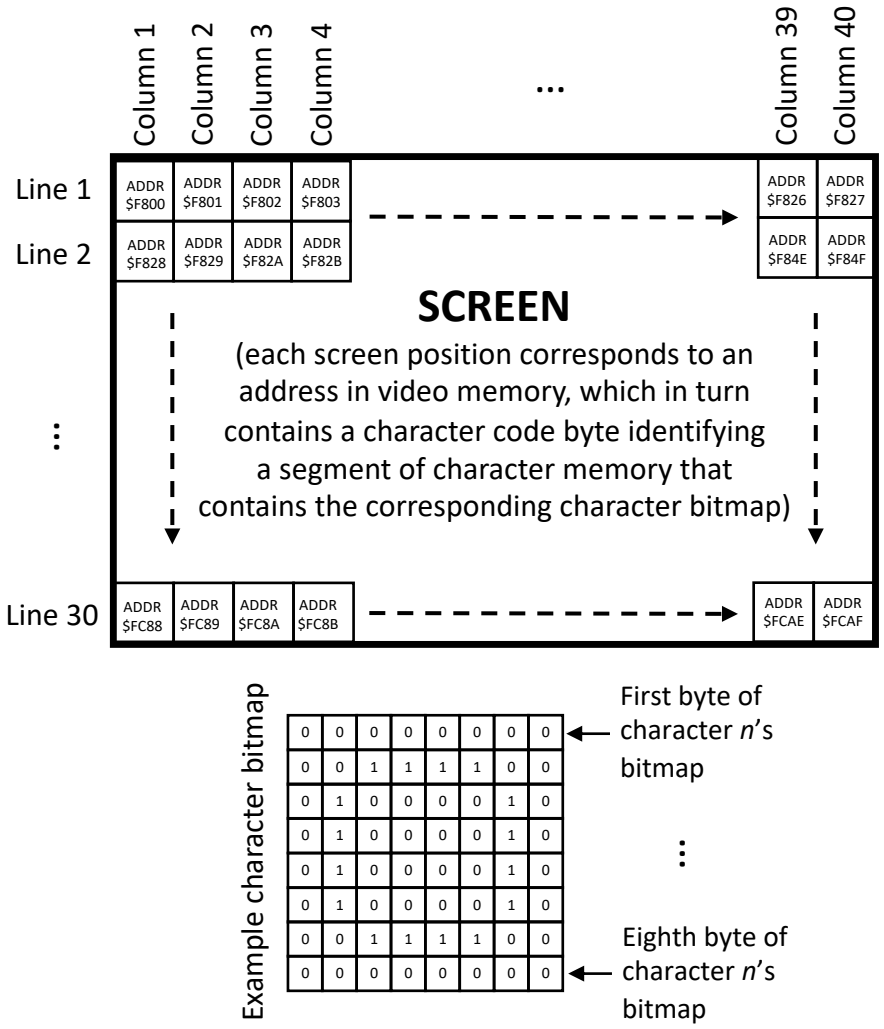
SPACER™ overview

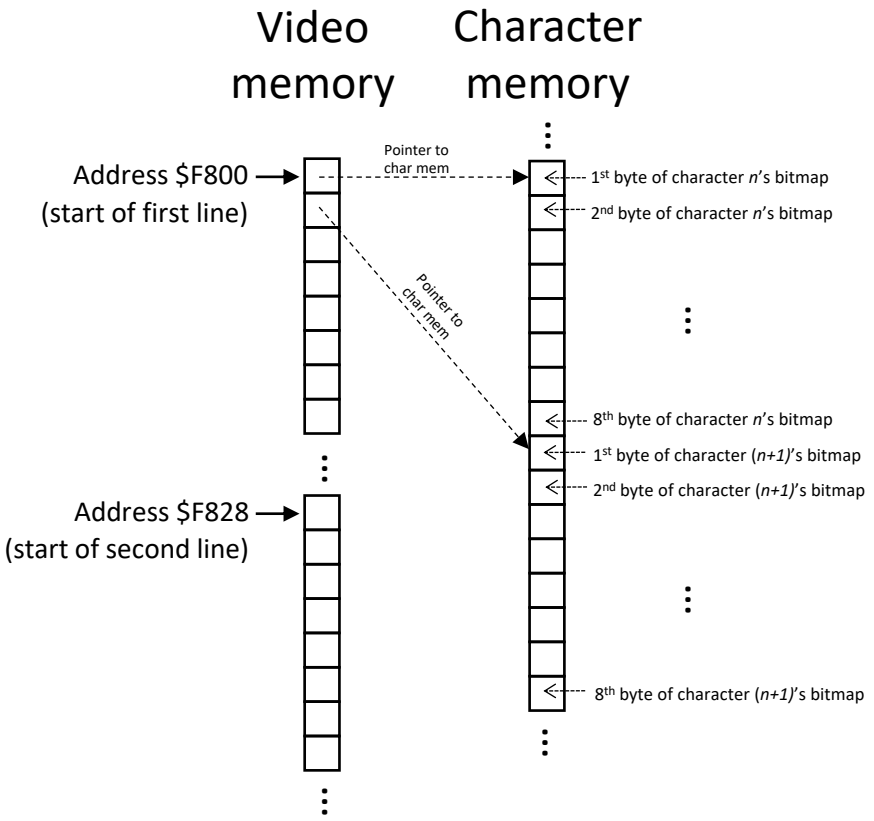
- SPACER is the glue that binds together the components of the *computer proper*
- Its design is the core of CERBERUS 2080's architecture, enabling the innovative multi-processor approach
- It replaces a standard control bus with a fully-connected control network that orchestrates the activity of the three processors and four memories
  - All control inputs and outputs of all ICs in the *computer proper* are processed by SPACER's internal logic
- SPACER's functions:
  - *Memory Selection*: enabling the appropriate memory IC depending on the contents of the address bus
  - *Read/Write control*: generating the appropriate write- and output-enable signals for the 4 memory ICs
  - *Clock Management*: generating and managing the clocks for CAT and the two CPUs
  - *CPU Control*: generating the signals to reset, start, halt, tristate and interrupt the CPUs, based on input from CAT
  - *Serial/Parallel Translation*: translating CAT's serial data and addresses into parallel bus accesses, and vice-versa

CERBERUS 2080™'s VGA mode

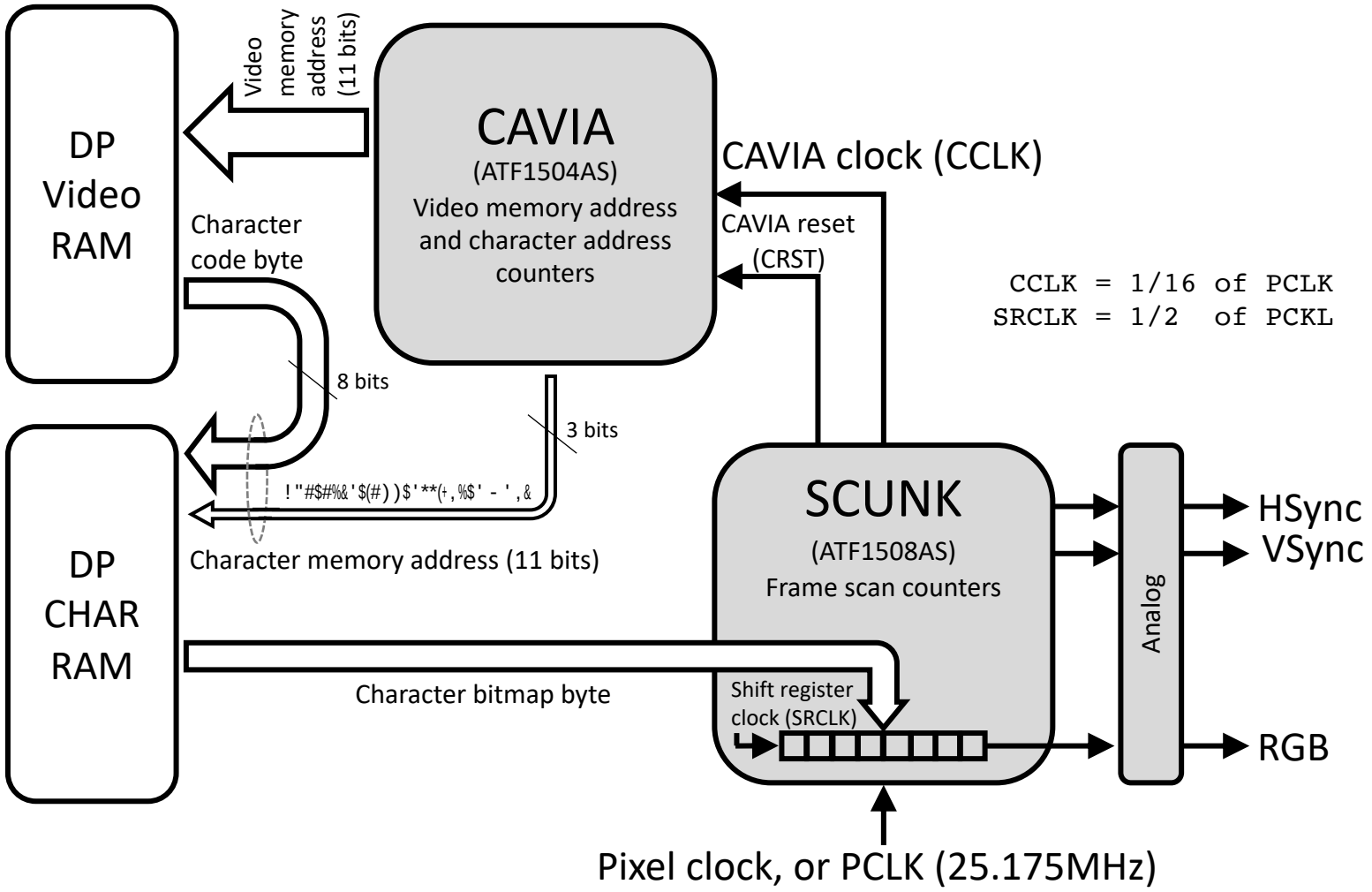
- CERBERUS 2080 uses a standard VGA mode with 640x480 *screen pixels* and a 25.175 MHz pixel clock
- But each *CERBERUS pixel* is a 2x2 matrix of *screen pixels*
  - CERBERUS 2080's resolution is thus 320x240 pixels, or 40x30 characters of 8x8 pixels each
- This translation is achieved by sampling each CERBERUS pixel *four times*
  - Twice horizontally and twice vertically
- Each of the 40x30 addresses in video memory holds a byte that identifies a particular character from the character set
- CAVIA reads out that byte, which then becomes the 8 most-significant bits of an 11-bit character memory address
- CAVIA reads out the 8 bytes of the corresponding character bitmap in character memory by progressively incrementing the 3 least-significant bits of the address from 0 to 7
- See sheets 10 and 11

Video and character memory organization





VGA circuit architecture



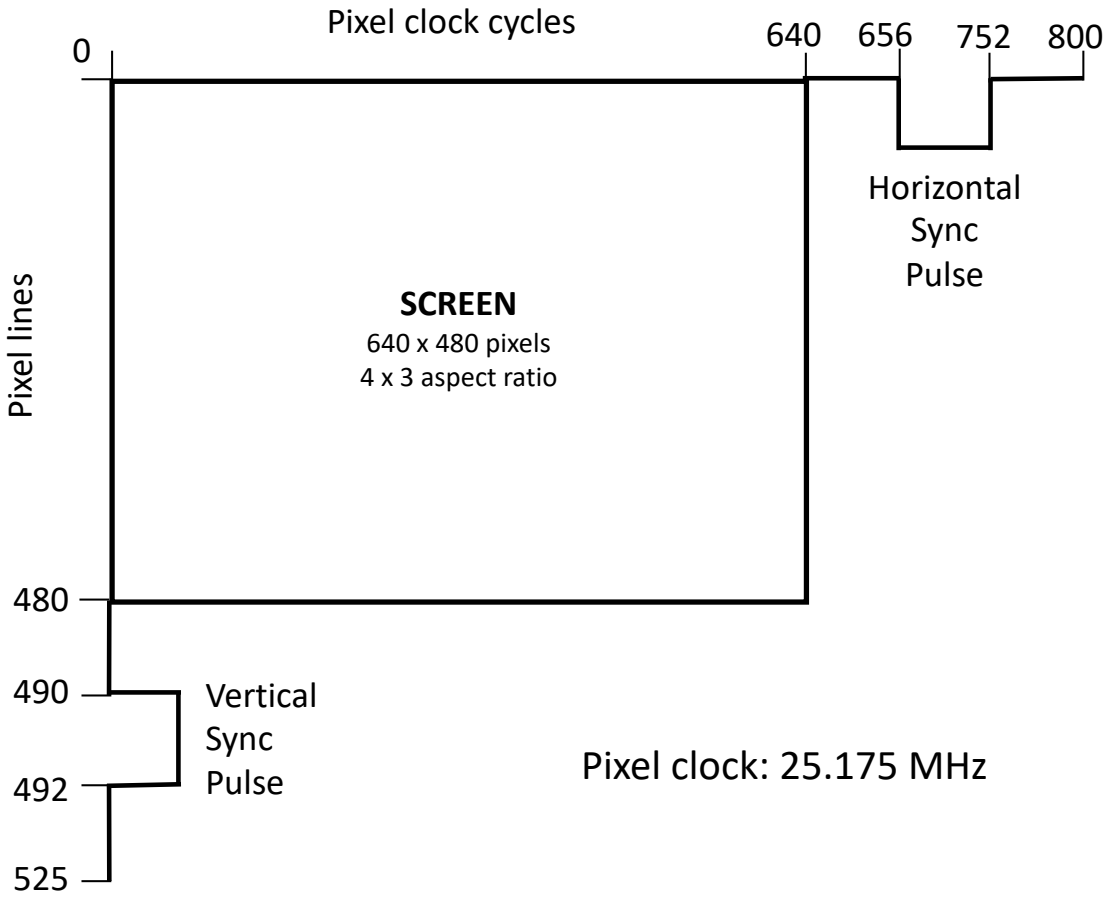
CAVIA™ overview

- CAVIA continuously scans all addresses of video memory, as well as the corresponding addresses in character memory
- Each 40-character line in video memory is scanned *16 times* per frame
  - 8 bytes per character times 2 passes per byte, so to double-sample it vertically
- Each read from video memory produces a character code byte that constitutes the 8 most-significant bits of the corresponding 11-bit character memory address
- CAVIA uses a counter to produce the 3 least-significant bits of the 11-bit address, so to scan all 8 bytes of the corresponding character bitmap in character memory
- Since each character memory read leads to 8 pixels (i.e. a byte from the character bitmap), which are then double-sampled horizontally (see sheet 13), CAVIA's clock (CCLK) is 1/16 of the pixel clock (PCLK)
- See sheet 11 again

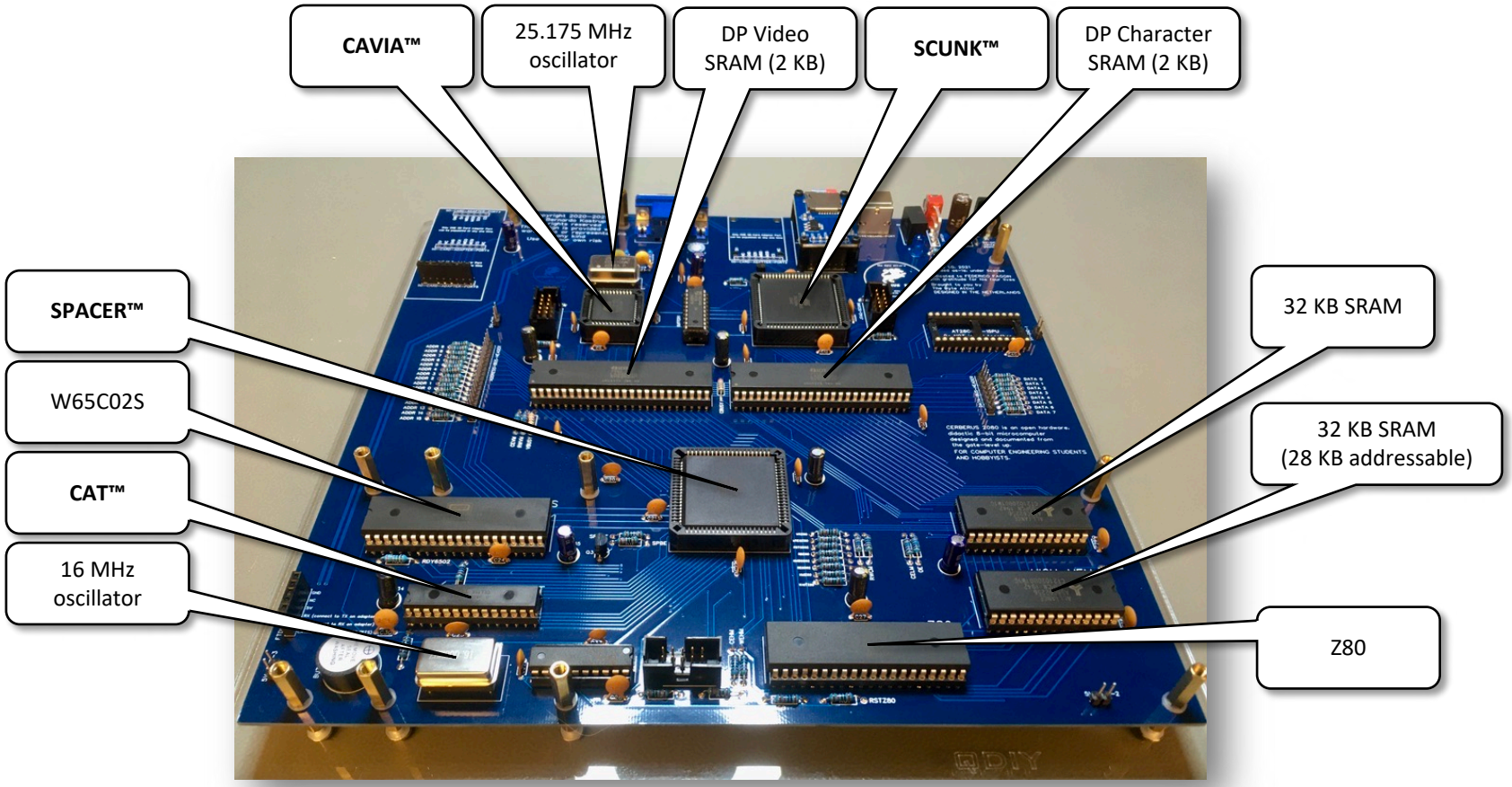
SCUNK™ overview

- Each video frame consists of *480 pixel lines* and is followed by a *vertical blanking interval* equivalent in timing to an additional 45 lines (see sheet 14)
- Each pixel line consists of *640 screen pixels* followed by a *horizontal blanking interval* equivalent in timing to an additional 160 screen pixels
- Both the vertical and horizontal blanking intervals encompass *synchronization pulses* with specific timings, as shown in sheet 14
- SCUNK is responsible for counting all the relevant intervals and producing the synchronization pulses
- It also generates the CAVIA clock (CCLK) from the pixel clock (PCLK) using an internal clock divider
- It features an internal 8-bit shift register, wherein each byte read out from character memory is temporarily stored (see sheet 11)
- It then shifts each bit of that byte out to the RGB line of the VGA connector, according to a shift register clock (SRCLK)
- SRCLK is generated internally by SCUNK, through a clock divider, and is ½ of PCLK so as to horizontally double-sample each bit in the shift register

Standard VGA signal



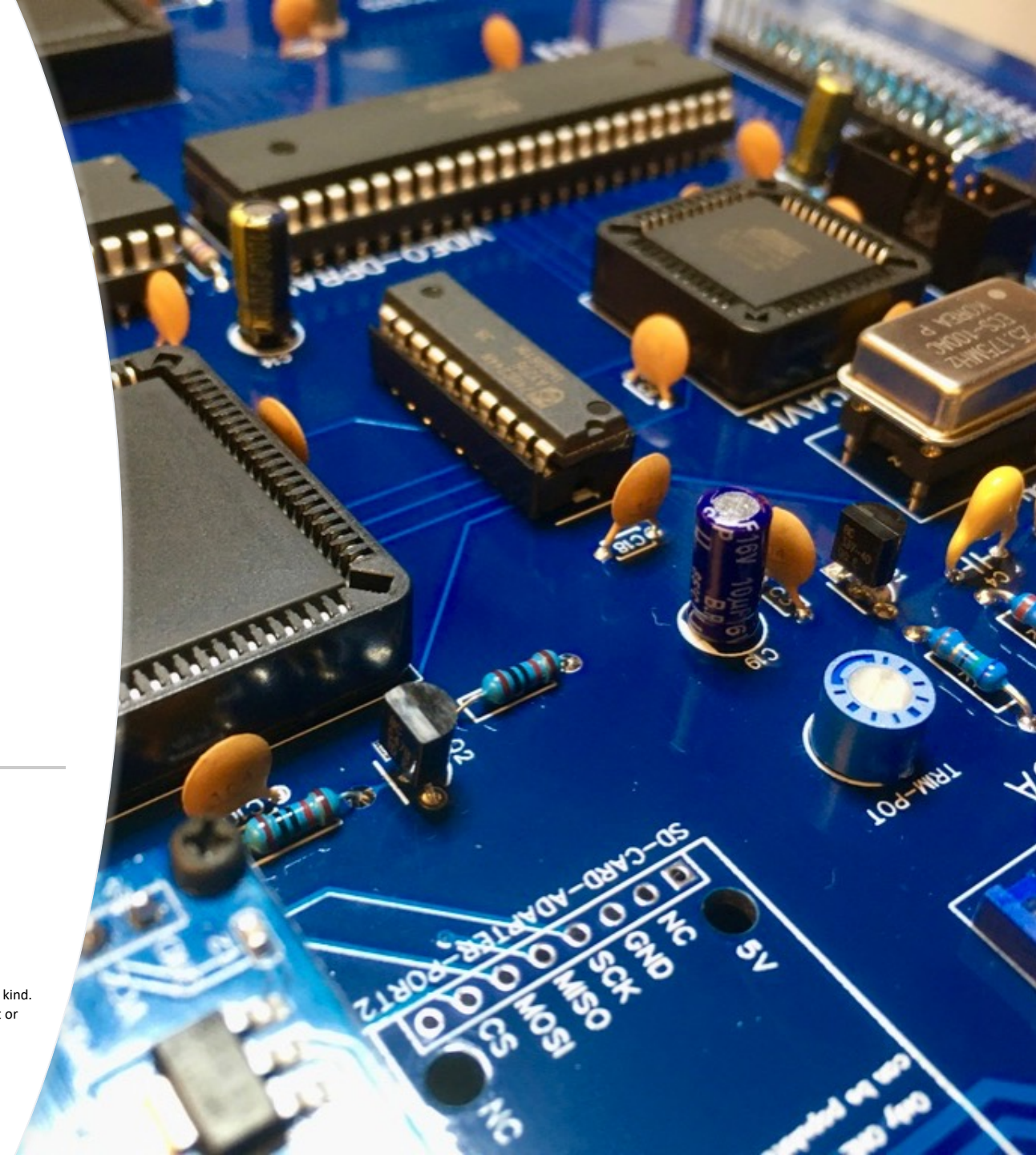
Board overview



The Byte Attic's

CERBERUS 2080™

Build Manual

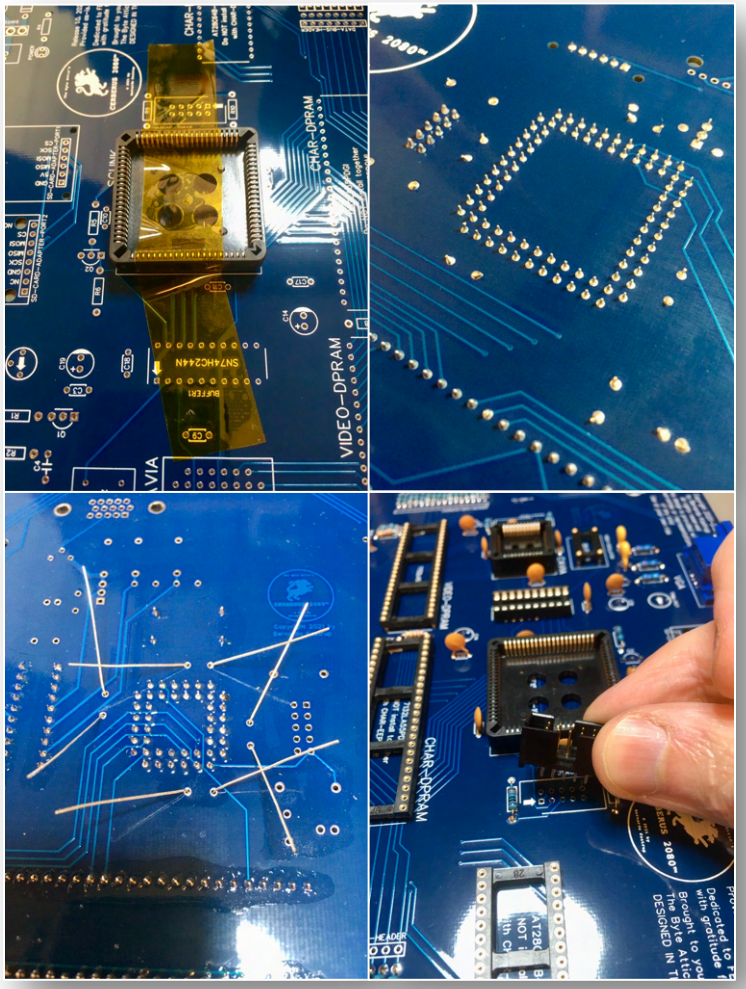


Building CERBERUS: preparation

1. ***Pay extra attention to avoid mistakes: it is virtually impossible to de-solder anything from CERBERUS's board, because of the metal planes***
2. Ensure you have all items in the Bill of Materials, the Bill of Materials itself and the Schematics
3. Carefully inspect your board for possible transport damage, such as scratches
4. Generously coat all pads on the backside of the board with *no-clean liquid flux*
   - You can use flux paste instead, but you will need to thoroughly clean the board afterwards
5. Make sure your soldering iron is at temperature before you start soldering
   - CERBERUS 2080's board has two large metal planes in the inner layers, which work as heat-sinks and increase the risk of cold solder joints
   - I recommend 350°C (662°F) for soldering, and 60/40 tin-lead solder wire
6. Beware: CERBERUS 2080's PCB is a modern board with *fairly small* pads
   - Use *just enough* solder to make a proper through-hole joint, for excess solder flows down the holes and can create shorts by forming invisible blobs under the sockets
   - I recommend a 1 mm (0.04") soldering tip, for larger tips will make it difficult to solder the PLCC sockets
7. Use a solder fumes extractor for your safety

Building CERBERUS: step 1

1. Start by soldering the sockets
   - Use tape on the frontside to keep them flush against the board while soldering, and *re-check* that they *are* indeed flush (you won't get a second chance)
   - Make sure to place the sockets in the *correct orientation*: the board is marked with arrows that indicate pin one
2. Solder the *ceramic* capacitors and resistors
   - Bend their leads on the backside to hold them flush while soldering
   - Cut the excess leads after soldering, *without bending the leads back*
3. Proceed to solder all connectors
   - VGA, PS/2, Switch, DC barrel and the 9 male pin-headers, all inserted on the frontside of the board
   - Cut the 40-pin pin-header into the appropriate sizes before soldering:
     1x 16-pin, 1x 8-pin and 4x 2-pin
   - Carefully observe the correct orientation of the 3 JTAG programming headers: pin one is marked with a small arrow on the board

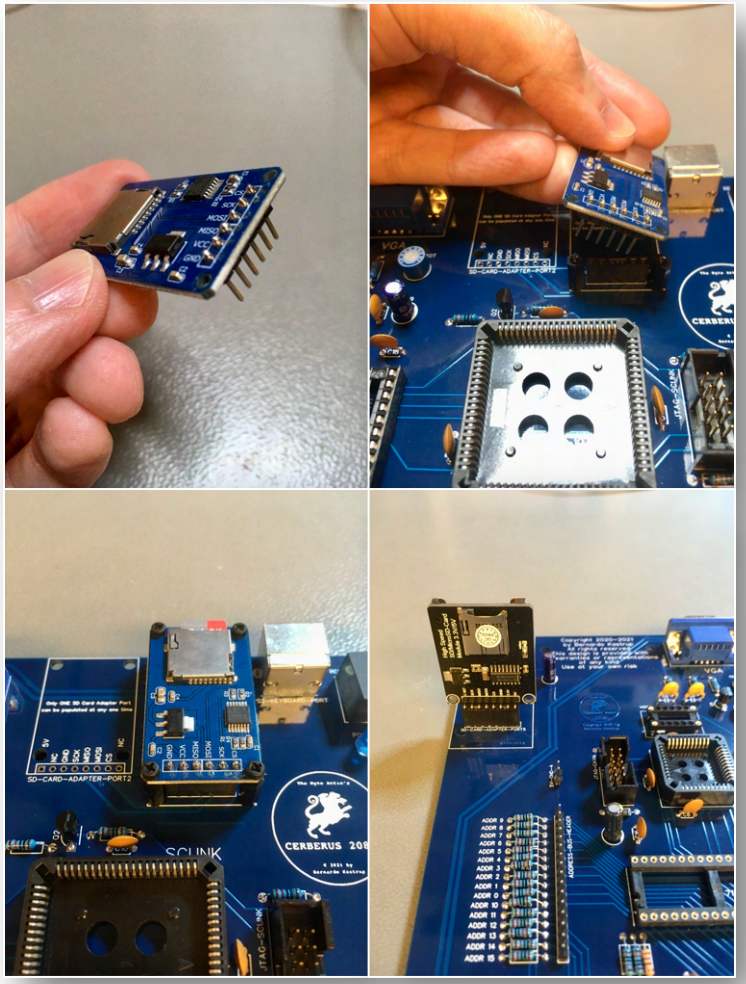


Building CERBERUS: step 2

1. Solder one MOLEX 90147-1106 on the SD-CARD-ADAPTER-PORT1 of the PCB
2. Choose an 8-pin SD card adapter port on the PCB to install the MOLEX 90147-1108
3. Install the other MOLEX 90147-1106 on the FTDI-ADAPTER-PORT of the PCB
4. Solder the trim-potentiometer, transistors, diode, LED and buzzer
   - *Observe the correct orientations*, as marked on the board!
   - Do *not* solder the diode flush against the board; keep it one or two millimeters above it, to allow for air circulation underneath it
5. Solder the DC-DC regulator, *observing the correct orientation* as marked on the board
6. Only now solder the electrolytic capacitors, as flush against the board as they will allow when gently pushed into their holes, but don't force them
   - Observe their polarity! The board has the correct polarity marked on it
   - Cut the excess leads after soldering

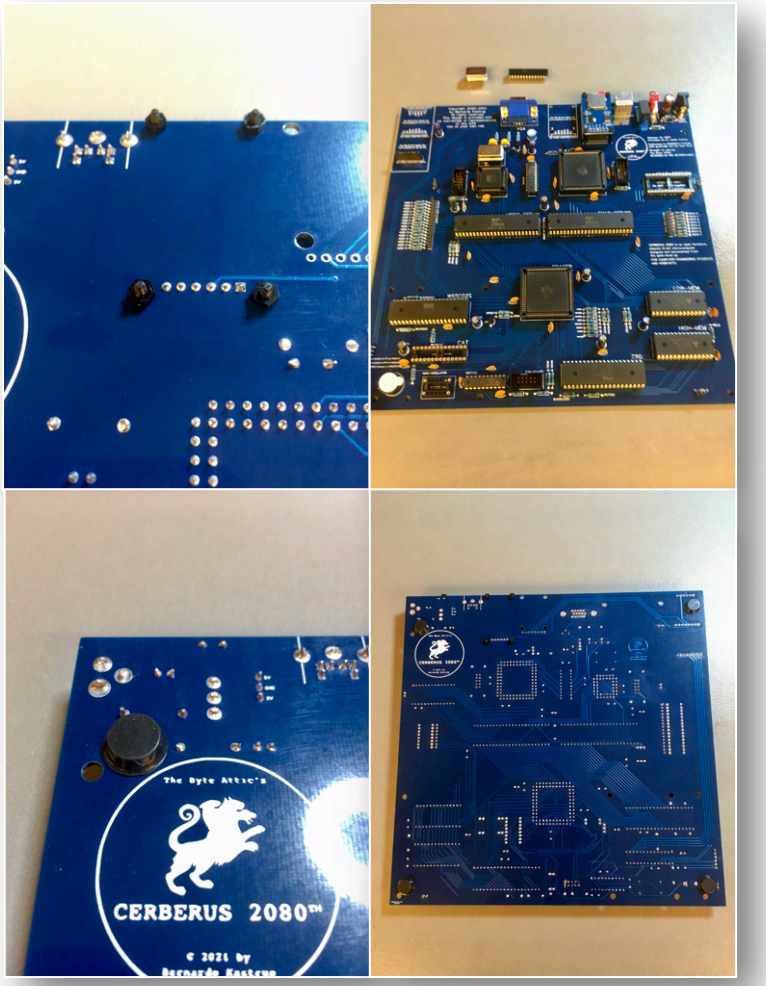
Building CERBERUS: step 3

1. If you are using the µSD card adapter listed in the Bill of Materials:
   - Carefully de-solder the male 6-pin pin-header that comes with it
   - Clean the now-open pads with IPA and apply no-clean flux
   - Cut a 6-pin vertical pin-header from what remains of the original 40-pin one
   - Solder the 6-pin vertical pin-header underneath the µSD card adapter, pointing downwards
   - Install the four male nylon stand-offs on the SD-CARD-ADAPTER-PORT1 of the PCB, nuts on the backside
   - Insert the µSD card adapter into the female receptacle on the board, and screw it on the stand-offs with matching nylon screws
2. If you are using a different (µ)SD card adapter, proceed analogously but using the matching SD card port on the board and matching stand-offs/nuts/screws
   - *Only one* SD-CARD-ADAPTER-PORT should be populated on the board!



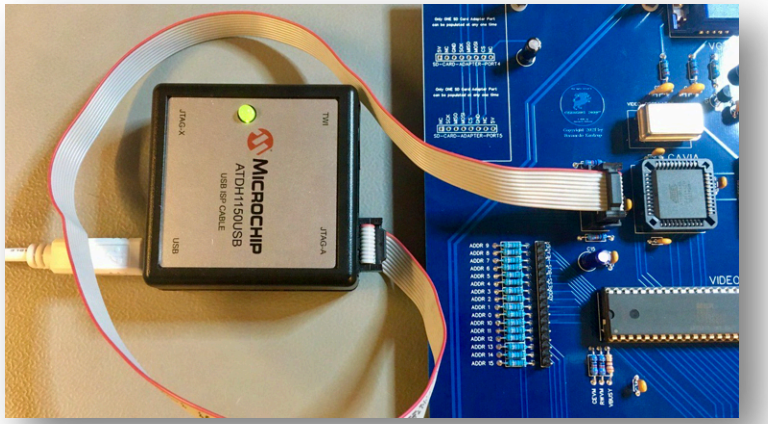
Building CERBERUS: wrap-up

1. Inspect *all* solder joints across the entire board, preferably with a magnifying glass, to check for shorts, cold joints, missing components or unsoldered leads
    - Make corrections as necessary
    - *Attention*: the CHAR-EEPROM position on the board does *not* need to, and should *not*, be populated, unless you are developing CERBERUS 2080 further and know what you are doing!
2. Clean the board carefully with IPA and ESD-safe tools, especially if you have used flux paste
3. Carefully populate the sockets with the appropriate ICs, *except CAT and the 16MHz oscillator*
    - Make sure all the pins are perfectly lined up before you push the parts into the sockets!
4. Apply the four rubber feet close to each corner of the board, on the bottom side, without the feet touching any solder joint



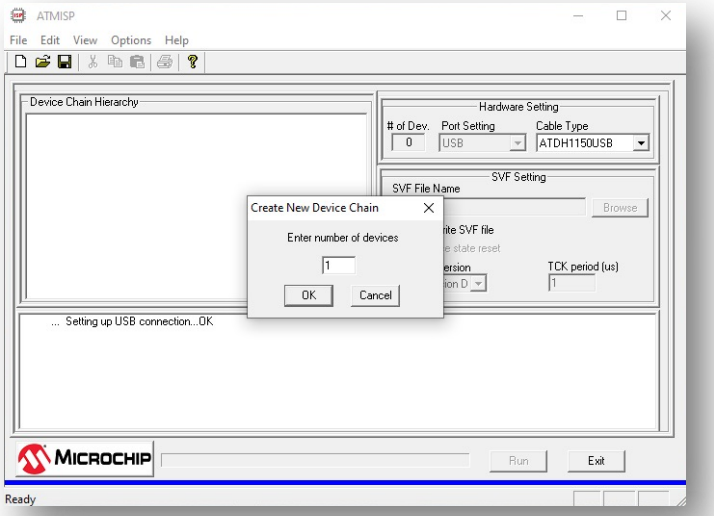
Programming the CPLDs: preparation

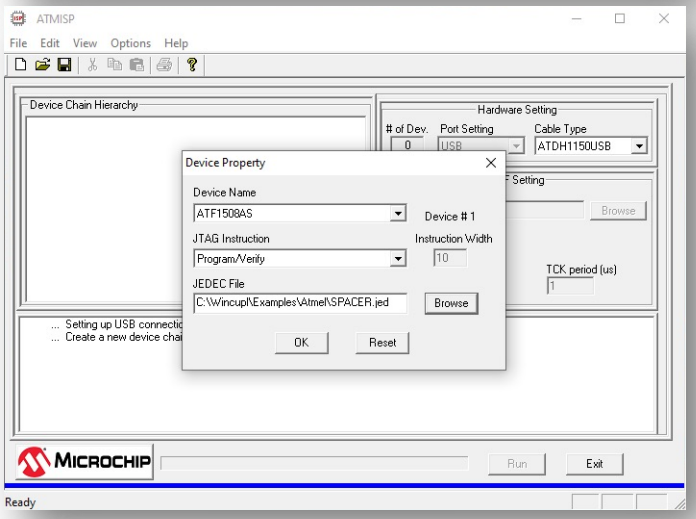
1. If you haven't purchased a pre-programmed chip set, you will need the following tools:
   - Microchip **ATDH1150USB** JTAG USB ISP cable
   - Microchip **ATMISP v7.3** (for Windows 7, 8 and 10), freely-downloadable in-system programing software: https://www.microchip.com/en-us/products/fpgas-and-plds/spld-cplds/pld-design-resources (also available in CERBERUS 2080's distribution)
2. Connect one end of the USB cable that accompanies the ATDH1150USB to the 'USB' port of the JTAG box, and the other end to your Windows PC
3. Connect one end of the 10-wire flat cable that accompanies the ATDH1150USB to the 'JTAG A' port of the JTAG box (see picture)



Programming the CPLDs

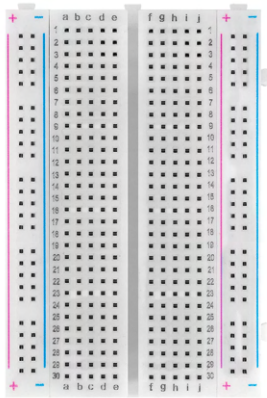
4. Connect the free end of the 10-wire flat cable to the appropriate JTAG header on the CERBERUS board (each of SCUNK, CAVIA and SPACER has a marked JTAG header close to it)
5. Connect CERBERUS to a 9V DC center-positive power supply capable of 500mA or more and turn it on (yes, *without* CAT and the 16MHz oscillator)
6. From the Windows computer, open ATMISP and click: `File → New`
7. In the pop-up window, enter `1` device and click `OK`
8. In the new pop-up window, under `Device Name` choose `ATF1508AS` for SPACER and SCUNK, or `ATF1504AS` for CAVIA
9. Under `JTAG Instruction` choose `Program/Verify`
10. Under `JEDEC File` click `Browse` and choose the appropriate .jed file from CERBERUS 2080's distribution (e.g. choose SPACER.jed to program SPACER), click `OK`
11. Click `Run` and wait for completion
12. Turn CERBERUS off
13. Repeat the above for SCUNK, CAVIA and SPACER, repositioning the 10-wire flat cable accordingly each time

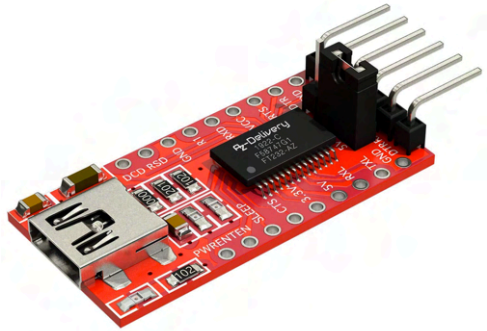




Programming CAT: preparation

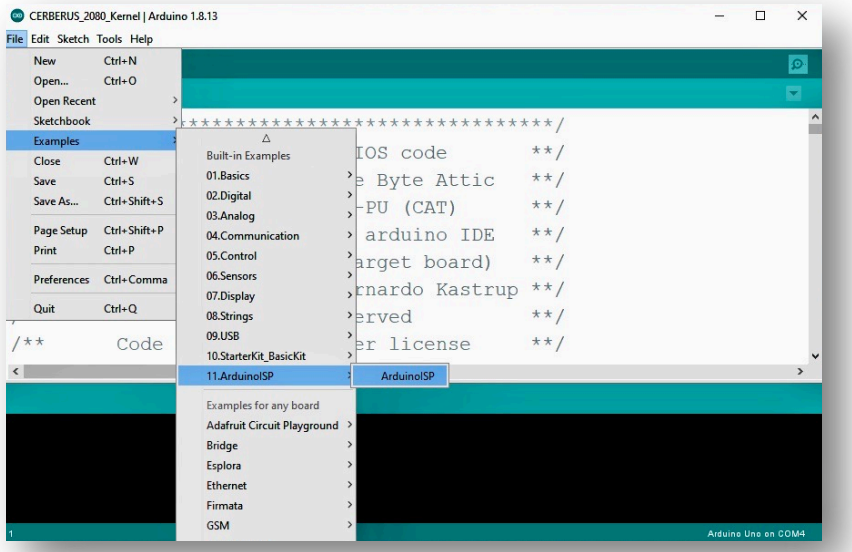
1. Unless you've purchased a pre-programmed CAT, you will need:
   - An Arduino UNO board with accompanying USB cable
   - An Arduino IDE installation on your PC: https://www.arduino.cc/en/software (also available in CERBERUS 2080's distribution)
   - An AZDelivery FTDI Adapter FT232RL USB to TTL Serial (or equivalent)
   - A standard USB to mini-USB cable
   - A small breadboard
   - Jumper wires
   - A 16MHz strong 4-pin oscillator (the one you'll use on CERBERUS)
   - A 10µF capacitor
   - A 10KΩ resistor





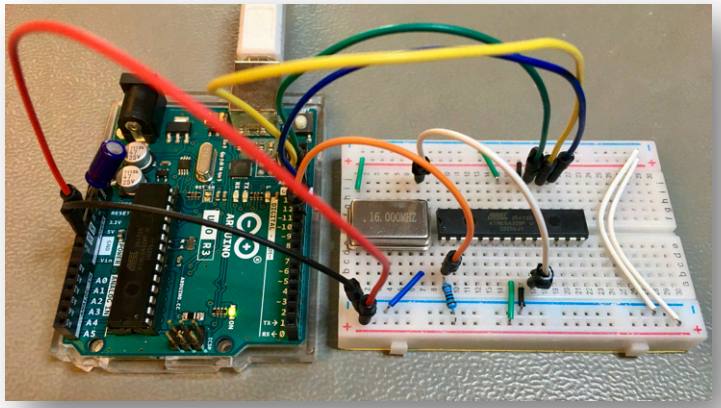
Programming CAT: changing the registry

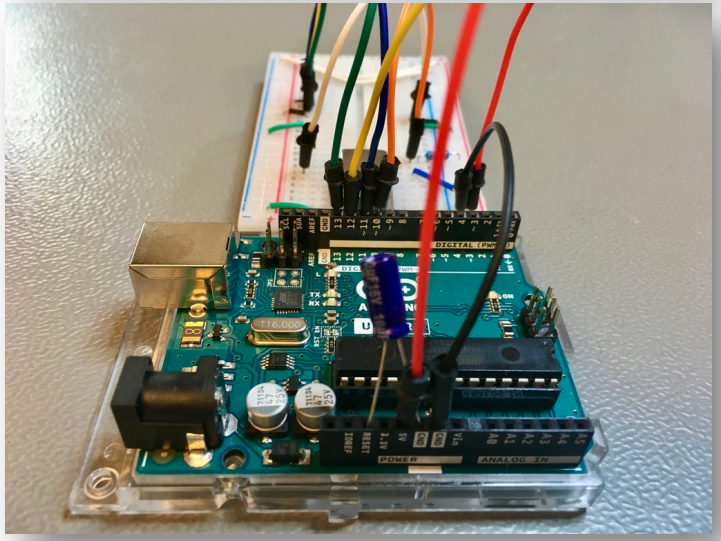
2. Connect the Arduino UNO to your computer (via USB) and launch the Arduino IDE
3. In the Arduino IDE load:
   `File → Examples → 11.ArduinoISP → ArduinoISP`
4. Compile and upload the ArduinoISP sketch to the Arduino UNO
5. When done, disconnect the Arduino UNO from your PC



Programming CAT: changing the registry

6. Set up the breadboard circuit as shown in the pictures
    - Connect the 10µF capacitor between GND and RESET of the Arduino UNO (watch out for the polarity!)
    - Connect GND and 5V from the Arduino UNO to the GND and 5V rails of the breadboard
    - Connect pin 1 of the ATmega328p to a 10KΩ pull-up resistor (itself connected to 5V) and to pin 10 of the Arduino UNO
    - Connect pins 8 and 22 of the ATmega328p to the GND rail of the breadboard
    - Connect pins 7 and 20 of the ATmega328p to the 5V rail of the breadboard
    - Connect pin 7 of the oscillator to the GND rail and pin 14 to the 5V rail of the breadboard
    - Connect pin 9 of the ATmega328p to pin 8 of the oscillator
    - Connect pins 17, 18 and 19 of the ATmega328p respectively to pins 11, 12 and 13 of the Arduino UNO
7. Reconnect the UNO to your PC





Programming CAT: changing the registry

8. Replace the file:
   `/Arduino/hardware/breadboard/avr/boards.txt`
   on your PC with the file `boards.txt` found in the `/CAT` directory of CERBERUS 2080's distribution and restart de Arduino IDE

9. From the Arduino IDE select:
   `Tools → Board → breadboard-avr (in sketch book) → ATmega328 16MHz external 4-pin oscillator`

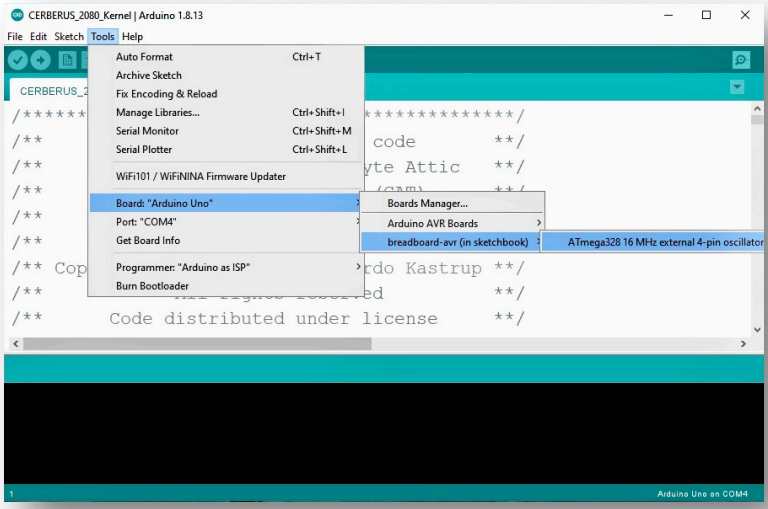
10. Still from the Arduino IDE change the COM port to the one active in your case:
    `Tools → Port →` (active port)

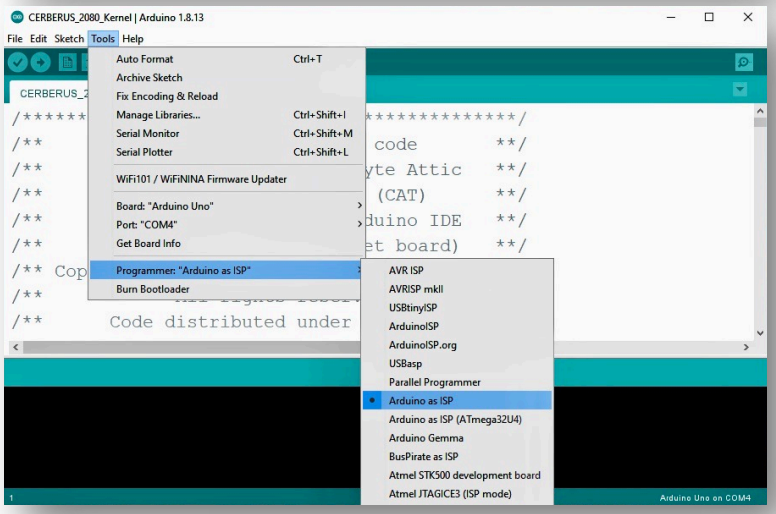
11. Now select:
    `Tools → Programmer → Arduino as ISP`

12. And finally:
    `Tools → Burn Bootloader`

13. After completion, disconnect the Arduino UNO from your PC

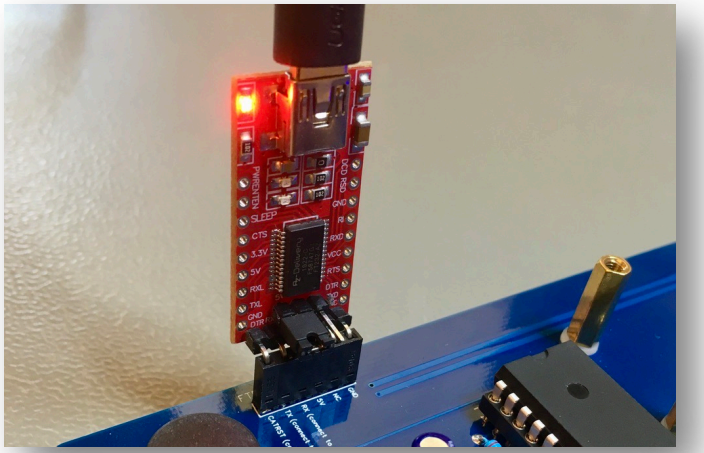
14. The registry is now changed! You may remove the ATmega328p from the breadboard and insert it into the appropriate socket on CERBERUS 2080's PCB, along with the 16MHz oscillator

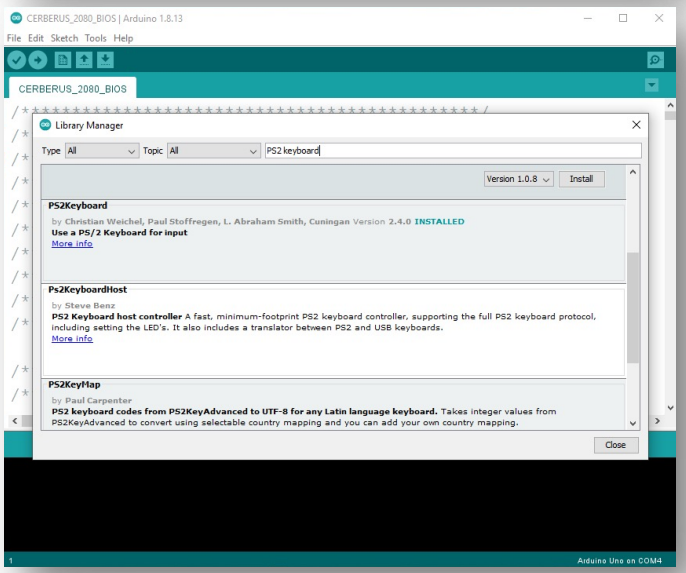




Programming CAT: uploading the BIOS

15. Set the voltage jumper on the FTDI adapter to 5V
16. Connect the FTDI adapter to your PC via the USB to mini-USB cable
17. Connect the FTDI adapter to the FTDI-ADAPTER-PORT of the CERBERUS board (make sure to match the signal markings on the two boards)
18. Turn CERBERUS on
19. Re-start the Arduino IDE
20. From the Arduino IDE, go to:
    `Tools → Manage Libraries`
21. In the pop-up window, do a search on 'PS2 Keyboard' and install the `PS2Keyboard` library by Christian Weichel, Paul Stoffregen et al.
22. From the IDE, select:
    Tools → `Board` → `Arduino AVR Boards` → `Arduino Uno`





Programming CAT: uploading the BIOS

23. Select the now-active COM port:
`Tools` → `Port` → (active port)

24. Change the programmer:
Tools → `Programmer` → `AVR ISP`

25. Load the `CERBERUS_2080_BIOS.ino` sketch available in the `/CAT` directory of CERBERUS 2080's distribution (you will be prompted to create a new folder, just accept and proceed)

26. Compile and upload the sketch (you may hear CERBERUS beep now, which is normal)

27. Disconnect the FTDI adapter from the CERBERUS board

28. Turn CERBERUS off

29. Load a µSD card with the CERBERUS µSD card files (see dedicated directory containing these files in CERBERUS 2080's distribution)

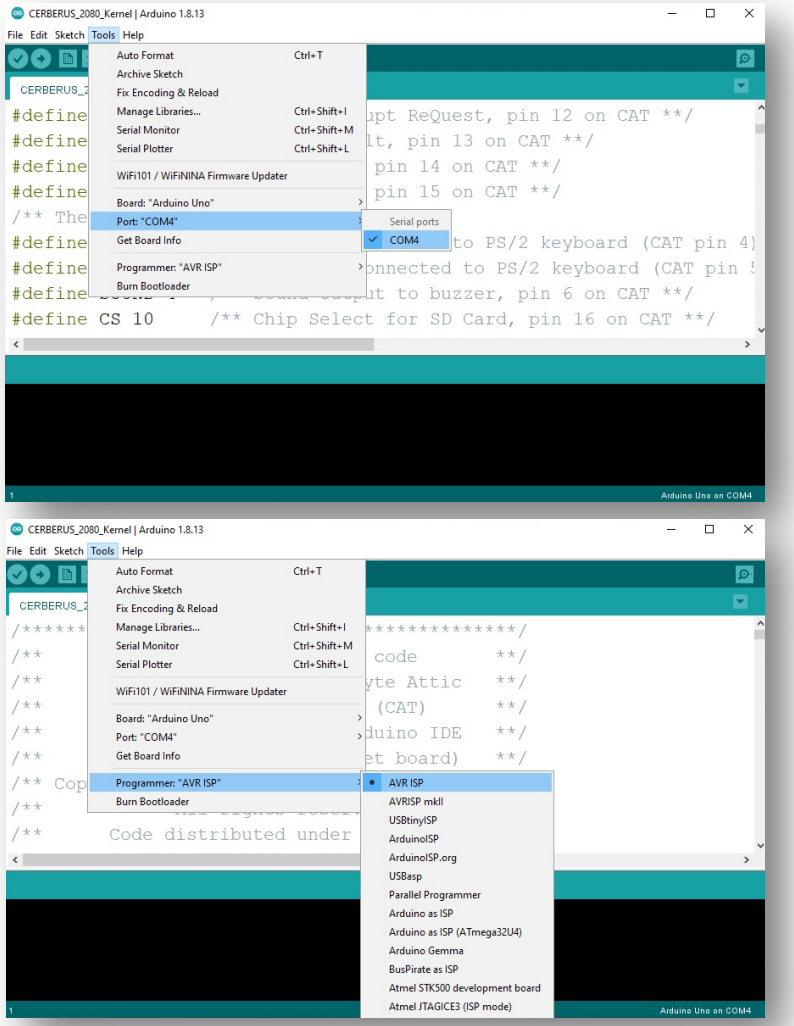
30. Insert the µSD card into CERBERUS's µSD card adapter

31. Connect CERBERUS to a VGA monitor, PS/2 keyboard and 9V DC center-positive power supply capable of 500 mA or more

32. Turn CERBERUS on

33. Turn the TRIM-POT all the way clockwise, with a small flathead screwdriver (the screen image will go dark); while observing the screen, slowly turn it anti-clockwise until the image brightness is to your taste, *but don't turn the POT beyond maximum brightness!*

34. You're done! CERBERUS 2080 should now boot normally and be fully functional



Optional: a transparent case

1. To use 2 transparent micro-ATX trays as case, in addition to the trays you will need:
    - 11x M3 20+6mm male metal stand-offs
    - 11x M3 10mm female metal stand-offs
    - 22x M3 matching metal screws
    - 44x M3 6x3x1mm plastic washers

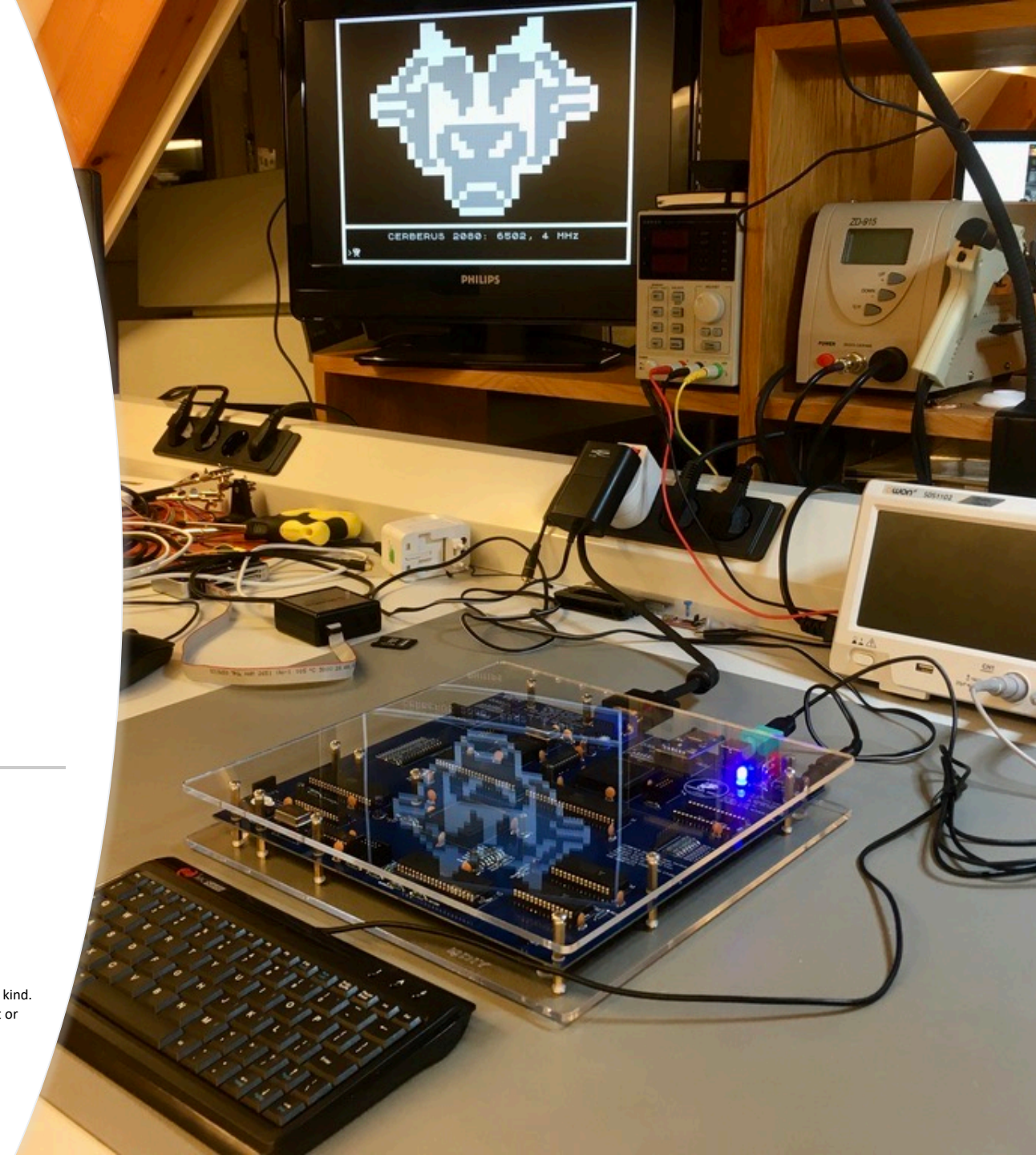
    (These are *not* included in the Bill of Materials!)
2. Place a plastic washer on a male metal stand-off
3. Put the male metal stand-off with the washer through one of the micro-ATX holes, *from the topside of the board*
4. Place another plastic washer on the other side of the hole, hanging from the threaded base of the male stand-off
5. Screw a matching female metal stand-off from the bottom-side of the board
6. Repeat for all 11 micro-ATX holes on the board
7. *Gently* screw the two acrylic trays onto the stand-offs, one tray on each side of the board (do *not* put too much pressure on the screws, for acrylic cracks easily)
    1. Put a plastic washer in between each screw and tray
    2. Acrylic dilates and contracts with temperature, so it may be fiddly to align all the holes. The CERBERUS board *does* have all micro-ATX holes at the correct locations

The Byte Attic's

CERBERUS 2080™

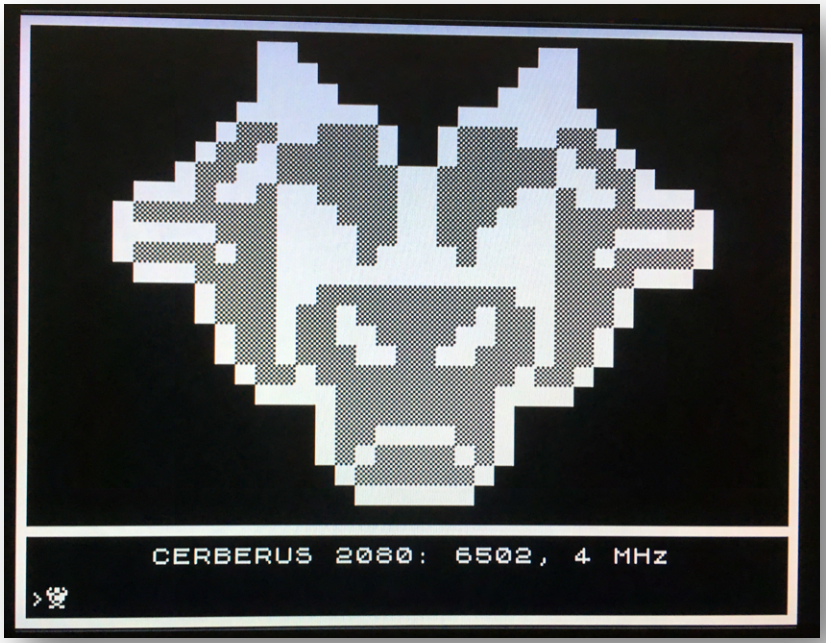
© 2021 by
Bernardo Kastrup

User's Manual



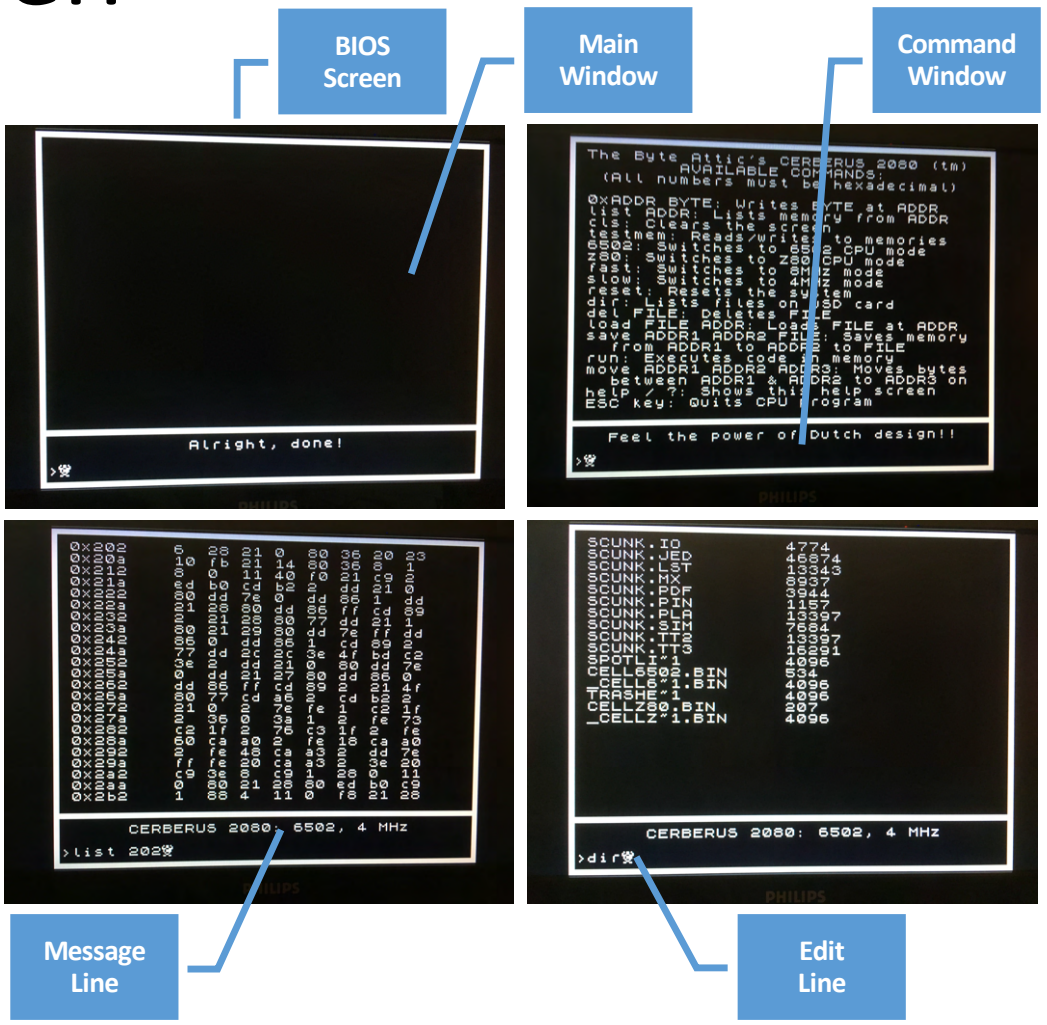
Powering CERBERUS 2080™ on

- To boot properly, CERBERUS needs to have a (µ)SD card inserted with two files in it:
  - `cerbicon.img`
  - `chardefs.bin`
- The first contains the 3-headed dog image displayed on the startup screen, and the second contains the system's default character definitions or bitmaps
- If CERBERUS finds no (µ)SD card, it will produce a repeating beep to indicate an error, and show garble on the screen
- If a (µ)SD card is found, but lacks the `chardefs.bin` file, CERBERUS will beep once and display garble
- If the `cerbicon.img` file is missing, CERBERUS will not produce a correct startup screen, but will otherwise be fully functional
- The correct startup screen is shown in the picture



The BIOS screen

- The user interacts with CERBERUS through the *BIOS Screen*, which is displayed upon startup
- The BIOS Screen has two segments: the *Main Window* above and the *Command Window* below
- The Command Window is further divided into two segments: the *Message Line* above and the *Edit Line* below
- CERBERUS displays system messages to the user in the Message Line
- The user gives CERBERUS commands by typing them out in the Edit Line
- Type "`help`" or "`?`" to display a list of the commands available, which will be discussed in more detail in the next pages



Commands: basics

- Typing the **`up-arrow`** fills the Edit Line with the last command typed
- Typing the **`down-arrow`** clears the Edit Line
- **`run`** executes the code currently in the code area (starting at address 0202 in hexadecimal) on the selected CPU
  - If the code in memory is meant for one of the CPUs, but you try to execute it on the other, the result will be unpredictable; reset or power cycle the system if this happens
- If the CPU is running code, type **`ESC`** to return to the BIOS Screen
- **`cls`** clears the Main Window
- **`reset`** resets CAT and the two CPUs, redraws the startup screen, but does *not* otherwise erase the contents of memory
- **`6502`** selects the W65C02S CPU
- **`z80`** selects the Z80 CPU
- **`fast`** sets the CPU clock at 8 MHz
- **`slow`** sets the CPU clock at 4 MHz
- **`help`** or **`?`** displays a summary of the commands available

Commands: file manipulation

- All numbers typed into the Edit Line are assumed by CERBERUS to be hexadecimal
  - Instead of typing '`0xFF`' or '`$FF`', type simply '`FF`'
- File names are *not* case-sensitive, so "`CODE.bin`" and "`code.bin`" will be the same file
- **`load FILE ADDR`** loads the contents of a binary file named `FILE` from the (µ)SD card into memory, starting from the address `ADDR` (in hexadecimal)
  - If `ADDR` is not provided, CERBERUS will default to 0202, the start of the code area
  - The file name `FILE` needs to be typed out in full
  - Example: "`load cell6502.bin`"
- **`save ADDR1 ADDR2 FILE`** saves the contents of memory from `ADDR1` to `ADDR2`, inclusive, to a binary file named FILE in the (µ)SD card
  - Example: "`save 202 2ff program.bin`"
- **`del FILE`** deletes the file named `FILE` from the (µ)SD card
  - The name `FILE` needs to be typed out in full
  - Example: "`del typez80.bin`"
- **`dir`** lists the files on the (µ)SD card
  - CERBERUS's file system, built into the BIOS, does *not* support sub-directories, so all files must be in the root of the (µ)SD card

Commands: editing memory

- **`list ADDR`** displays the contents of memory from address ADDR onwards, until the Main Window is filled up
  - If `ADDR` is not provided, CERBERUS will default to `0000`
  - Example: "`list F800`"
- Again, all numbers typed into the Edit Line are assumed by CERBERUS to be hexadecimal
  - Instead of typing '`0xFF`' or '`$FF`', type simply '`FF`'

- **`0xADDR BYTE(s)`** inserts the list of bytes `BYTE(s)` (wherein individual bytes are separated by spaces or commas) into memory, starting from address `ADDR`
  - The list of bytes can be as long as it fits in the Edit Line, or have a single byte
  - You can use this command to write directly into video and character memories, which is handy to alter and see the character definitions
  - Try out these cool examples (after "`cls`"):
    "`0xF82A 00 01 02 03 04 05 06 07`"
    "`0xF000 FF 00 FF 00 FF 00 FF 00`"

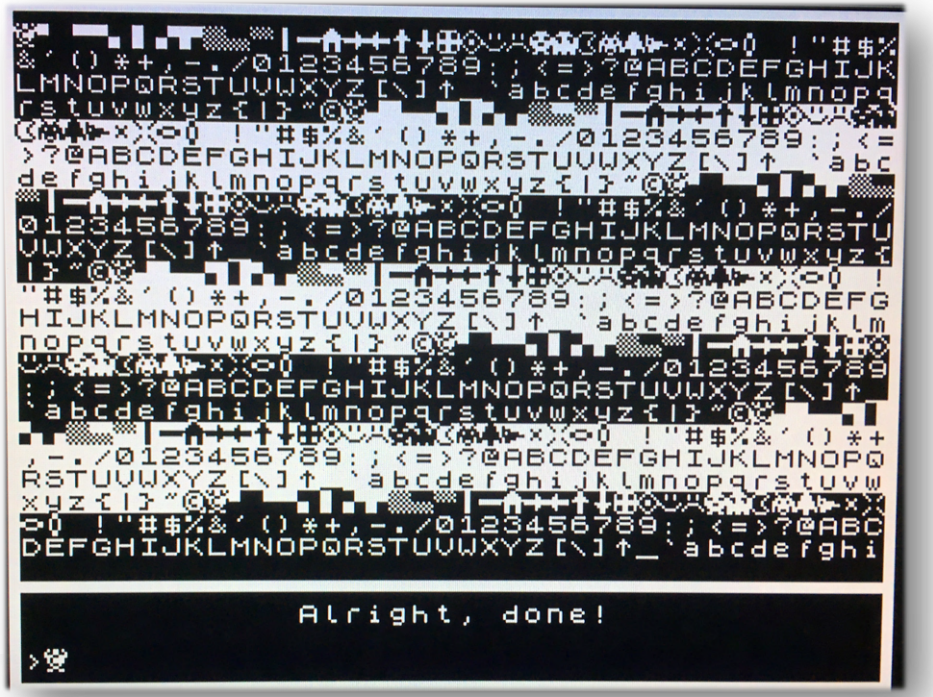
Commands: editing memory

- **`move ADDR1 ADDR2 ADDR3`** copies the contents of memory in the segment between `ADDR1` and `ADDR2`, inclusive, to the segment starting at `ADDR3`
  - Example: "`move 0000 00FF FF00`"

- **`testmem`** writes a sequence of numbers into low memory, then reads it from low memory and writes it to high memory, then reads it from high memory and writes it to video memory, where the sequence is interpreted as characters and displayed
  - This is a *non-exhaustive* test of the memory subsystem
  - It also displays the character set, as currently defined in character memory, on the screen

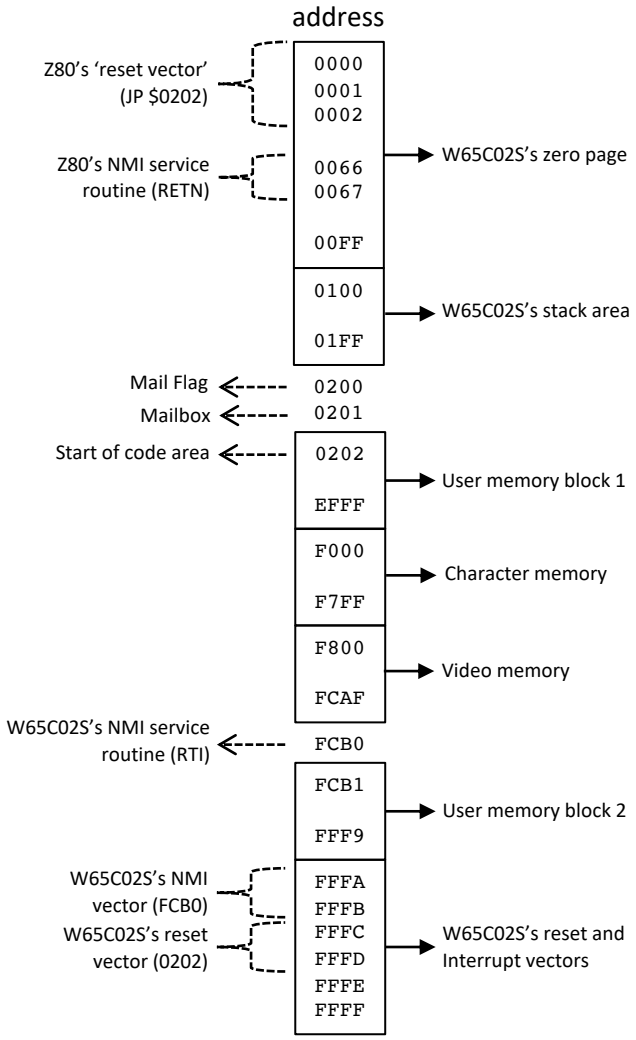
The character set

- CERBERUS allows for 256 different character definitions, each identified by an extended ASCII value, or character code, ranging from 0 to 255
- The picture illustrates the default character set, listed from 0 to 255 from the top left (it repeats itself after it ends)
- The character definitions are stored in addresses F000 to F7FF (inclusive)
- Each character is an 8x8 bitmap, and thus requires 8 bytes in its definition
- The bitmap of character 0 (the cursor character) is stored in addresses F000 to F007
- The bitmap of character 1 is stored in addresses F008 to F00F, and so on
- All character definitions can be modified by the user from the BIOS Screen or dynamically, from a program running on one of the CPUs
- Try this command, which changes the cursor:
  `"0xF000 FF 00 00 00 00 00 00 FF"`
- To reload the standard definitions again, type:
  `reset`
  or
  `load chardefs.bin F000`



The memory map

- The picture shows CERBERUS's memory map
- Areas of notice (all address are in hexadecimal):
  - Address 0202 is the start of the code area for both CPUs
  - Address 0200 is the *Mail Flag*: when a CPU is running and CAT (running the BIOS) detects a key press, it pokes a 1 into this address
  - Address 0201 is the *Mailbox*: when a CPU is running and CAT (running the BIOS) detects a key press, it pokes the character code of the pressed key into this address
  - Any byte poked into the video area will be interpreted as a character code and the corresponding character bitmap will be displayed on the screen
  - Any byte poked into the character memory will be interpreted as an update to a character definition
  - The largest contiguous area available for programs and data is the user memory block 1, which resides physically in the two 32KB SRAM ICs
  - The user memory block 2 is also available, and resides physically in the DP Video SRAM



Example applications

- CERBERUS 2080's distribution contains 4 example applications:
  - TYPE6502.bin
  - TYPEZ80.bin
  - CELL6502.bin
  - CELLZ80.bin
- The corresponding source (.asm) and hexadecimal (.hex) files of the 'CELL' application are also provided in the distribution
- The files with '6502' in their names are meant for the W65C02S CPU
- The files with 'Z80' in their names are meant for the Z80 CPU
- The 'TYPE' application simulates a typewriter on the screen: it prints out, in sequence, the keys you press on the keyboard
  - It tests the interface between CAT (which reads the keyboard) and the active CPU (which receives the character code read via memory-mapped I/O) during application execution
- The CELL application is a Wolfram Rule-30 linear cellular automaton that fills the screen with semi-random vertical-scrolling patterns
  - It tests the speed of CERBERUS's software-based scrolling
- The CELL6502.bin code is slightly more sophisticated then the CELLZ80.bin code
  - The W65C02S version also updates the character definitions in the character memory on-the-fly, during execution
- The BIOS will not prevent you from trying to run W65C02S code on the Z80, or Z80 code on the W65C02S
  - If you accidently do so, the result is unpredictable, although most of the times nothing happens
  - I recommend resetting or power cycling CERBERUS if you do this accidentally

Tips and tricks

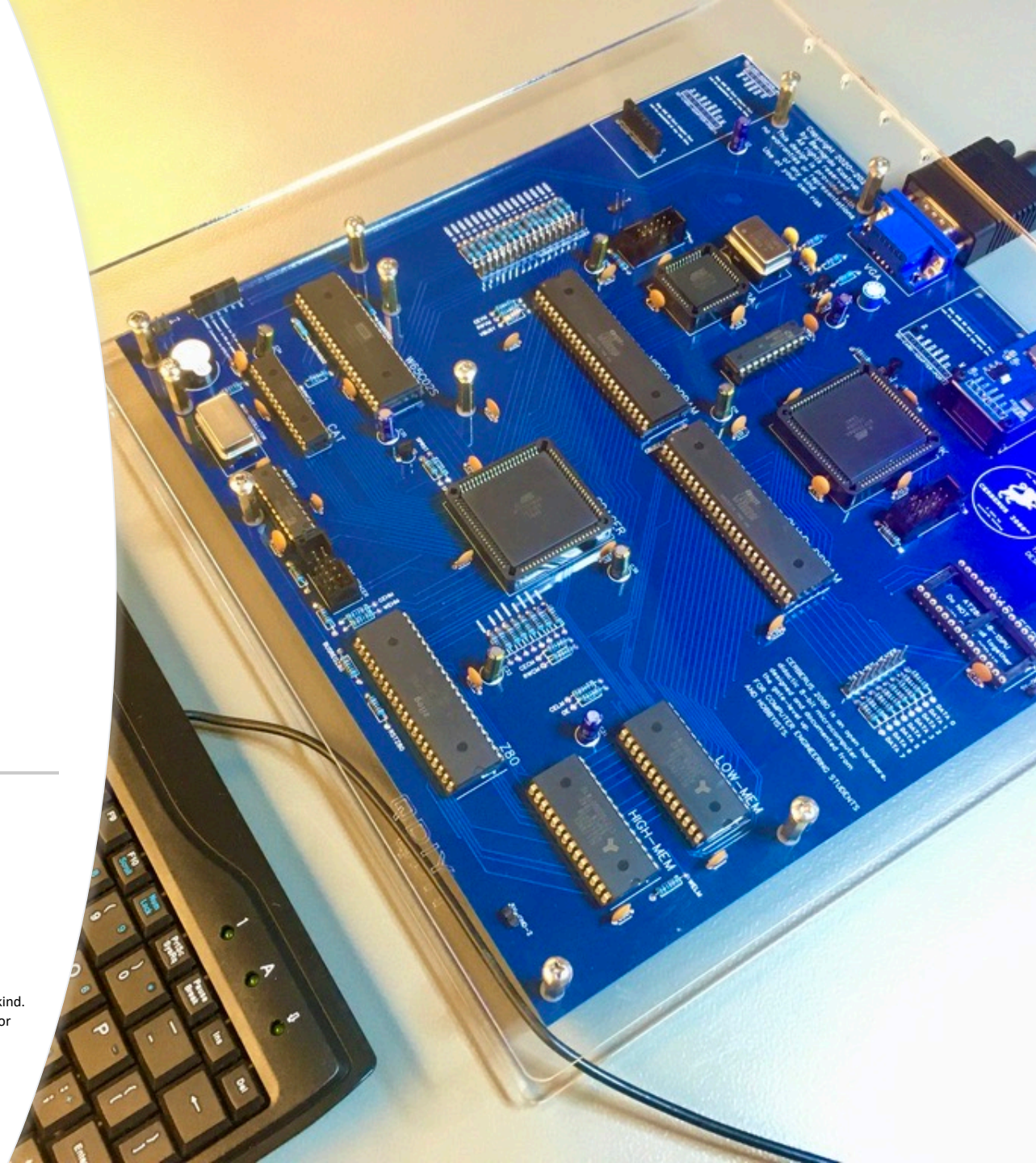
- You can load new programs by turning CERBERUS off, removing the (µ)SD card and loading new files into it from your PC
- Remember that you can control the CPU clock speed from the BIOS Screen: if a program is running too slow or—which is more likely—too fast, you can partly compensate for it with the **`slow`** and **`fast`** commands
- Clean the CERBERUS board only with ESD-safe brushes and/or ESD-safe compressed gas
- There are tiny, modern USB keyboards—such as the *MC Saite*—which can be used with CERBERUS with a simple USB-to-PS2 adapter
- Monitor updates to the CERBERUS distribution files for firmware improvements and eventual bug fixes
- Since SPACER, SCUNK and CAVIA are CPLDs, even hardware updates to the heart of CERBERUS can be performed without physical modifications to the board

The Byte Attic's

CERBERUS 2080™

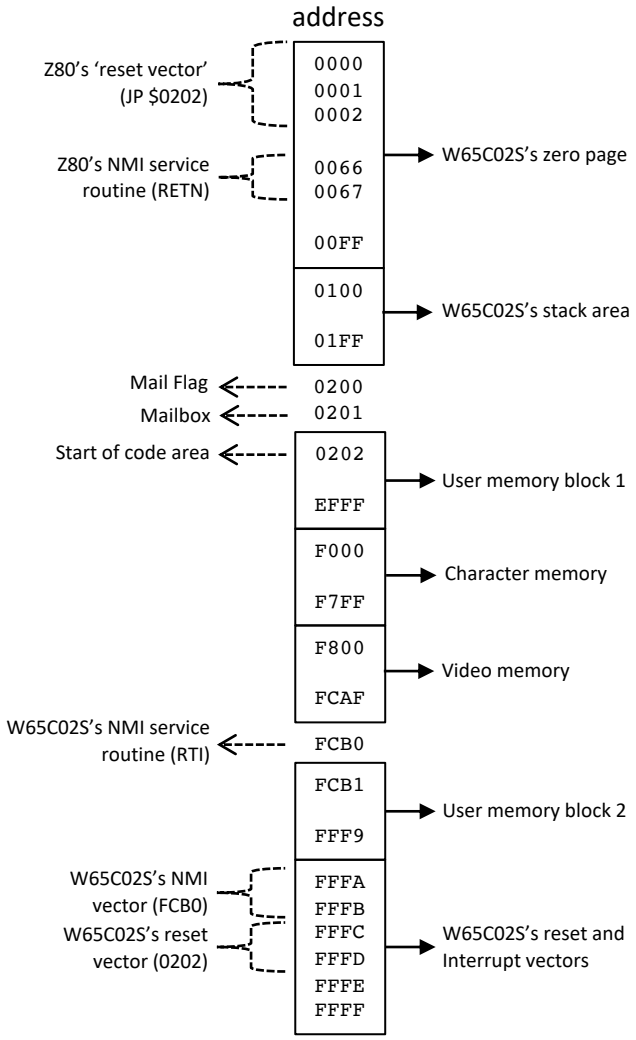
© 2021 by
Bernardo Kastrup

Programmer's Notes



More memory map considerations

- The picture shows CERBERUS's memory map
- The BIOS will automatically put a `JP $0202` on addresses 0000, 0001, and 0002 to simulate a reset vector for the Z80
- The BIOS will automatically put a simple `RETN` (return) instruction on addresses 0066 and 0067, where the Z80 looks for its non-maskable interrupt service routine
- All I/O is memory-mapped to addresses 0200 (the Mail Flag, which signals the presence of new keyboard input) and 0201 (the Mailbox, which stores the character code of the last key pressed)
- Address 0202 is the start of the code area for both CPUs
- The BIOS will store an `RTI` (return from interrupt) on address FBC0, which is the non-maskable interrupt service routine for the W65C02S
- The BIOS will ensure that the W65C02S's NMI and Reset vectors are set according to the above
- The largest contiguous area of memory available for code and variables is the user memory block 1
- User memory block 2, physically residing in the DP Video SRAM, is also available to programmers
- Z80 coders can use the address space of the W65C02S's zero page and stack area as regular memory (except for addresses 0000 to 0002, 0066 and 0067, which are reserved)
- Z80 coders can also use the top 6 addresses as regular memory (which are otherwise reserved for the W65C02S's reset and interrupt vectors)



Physical memory map

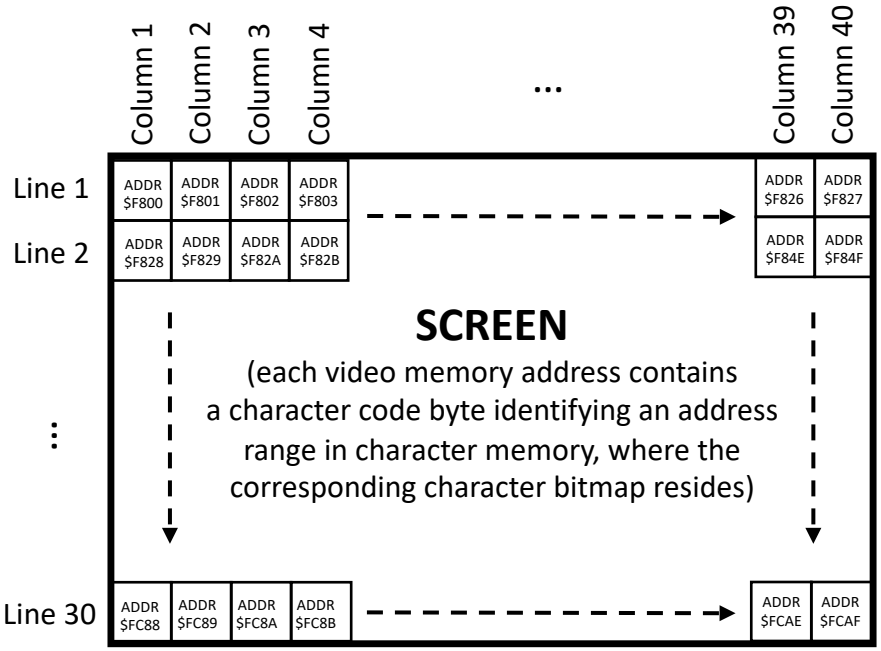
Low memory (32KB SRAM IC)

High memory (32KB SRAM IC)

Character memory (2KB DP SRAM IC)

Video memory (2KB DP SRAM IC)

Video memory organization



Notes to programmers

- CERBERUS 2080 supports two instruction sets: the Z80's and the W65C02S's
  - But applications must target one *or* the other CPU, not both concurrently
- CAT handles keyboard inputs when a CPU is running an application
  - Whenever a key is pressed, CAT halts the CPU, pokes a 1 into the Mail Flag address, pokes the corresponding character code into the Mailbox address, releases the CPU, and then issues a non-maskable interrupt (NMI)
- The BIOS code defines the non-maskable interrupt service routines as mere returns from interrupt (`RTI` for the W65C02S and `RETN` for the Z80)
  - An application running on a CPU must read keyboard inputs by *polling* the Mail Flag to see if there is a new input from the keyboard, and then reading out the inputted character code from the Mailbox
- Applications should always clear the Mail Flag by poking a 0 into it immediately upon reading out the Mailbox
- If you want to use more elaborate NMI service routines for keyboard input—instead of Mail Flag polling—your initialization code should use load (Z80) or store (W65C02S) instructions to put the corresponding opcodes/bytes in addresses 0066 (Z80) or FCB0 (W65C02S) onwards
- Since CAT also issues a non-maskable interrupt upon updating the Mail Flag and Mailbox, wait-for-interrupt instructions (`WAI` for the W65C02S, `HALT` for the Z80) are supported
- Maskable interrupts are *not* supported
  - Non-maskable interrupts are edge-triggered and, therefore, much simpler to handle in the hardware
  - CERBERUS is an educational platform unlikely to be used for real-time applications, so the lack of maskable interrupts shouldn't be a problem
- The Z80's port instructions (such as `IN` and `OUT`) are *not* supported
  - All I/O is memory-mapped, through the Mail Flag and Mailbox
- Applications can*not* use special characters (such as the arrow keys) as inputs, only letters and numbers
- The buzzer is only available to CAT, not to the CPUs
- The .asm and .hex files of example applications are provided in CERBERUS 2080's distribution, which illustrate how to deal with Mail Flag polling and wait-for-interrupt instructions in both CPUs
- I also recommend that programmers writing applications for CERBERUS acquaint themselves with the BIOS code, available in CERBERUS 2080's distribution

Assembler considerations

- Your code should *always* start at address 0202 (in hex), *even for the Z80*
  - The BIOS will put a JP $0202 on address 0 to simulate a reset vector
  - You should tell the assembler that your code starts at $0202
- CERBERUS's loader is very simple: it merely reads the binary file's bytes and stores them sequentially in memory, from address 0202 onwards
  - Only addresses and opcodes should be part of the assembled machine code!
  - No loader directives are supported
- See the files in the folder

  `/CERBERUS Applications Source Code`

  for examples of assembly programs and corresponding hex files for both CPUs
- If you want to change character definitions as part of your code, you should do it with load (Z80) or store (W65C02S) assembly instructions, as CERBERUS's loader will not recognize address directives

Tips and tricks

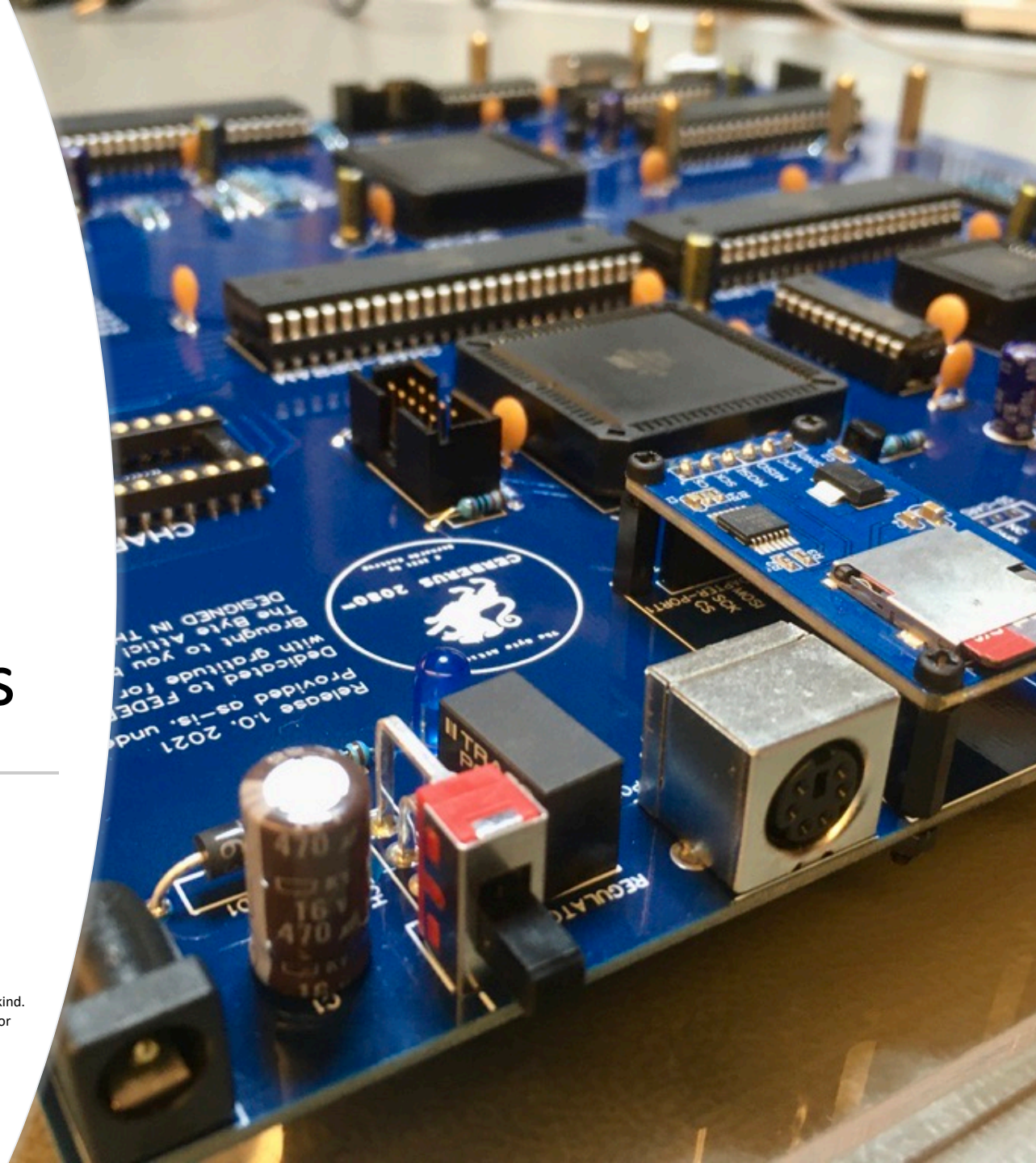
- CERBERUS 2080 is more-than-fast-enough to do software scrolling
  - Particularly in the Z80 CPU, with instructions like `ldir`
- Since its character set is re-definable on-the-fly, *smooth, pixel-wise* software scrolling should also be possible with appropriate redefinitions of the relevant characters
  - CERBERUS should have enough performance for this to happen seamlessly
- Since CERBERUS renders the entire memory accessible without restrictions, programmers can 'overrule' the BIOS by loading (Z80) or storing (W65C02S) opcodes and data bytes into memory locations normally controlled by the BIOS, without having to change the BIOS
  - A programmer can, for instance, re-write the NMI service routines from their own application code
- Porting the existing Z80 and 6502 code base to CERBERUS should be relatively easy, which opens up an opportunity for a lot of 'low-hanging fruit,' quick-reward development work

The Byte Attic's

CERBERUS 2080™

© 2021 by
Bernardo Kastrup

Electrical design considerations



Power supply and signal integrity

- The CERBERUS board operates at a single voltage: 5V DC
- It must be fed by a 9V DC, center-positive, external power supply rated for 0.5A or higher
- An on-board DC-DC regulator (Traco Power) produces the internal 5V from the 9V of the external power supply
- CERBERUS's maximum current consumption actually hovers just under 350 mA
  - At start-up, however, there is a normal current surge until all capacitances are charged, so the DC-DC regulator and safety diode are rated well above 350mA (namely, 2A, which is admittedly overkill)
  - The board also features four Vcc-GND pin-headers that can be used to power development breadboards connected to CERBERUS, and this may also increase the current draw
- Unlike linear voltage regulators, the DC-DC regulator hardly produces any heat
- The CERBERUS board has four layers and features GND and Vcc planes in the two inner layers, which work as a large, distributed capacitor
  - Helps with decoupling and bypassing
- Most of the capacitors are also meant to enhance decoupling and bypassing, so to ensure a very stable power signal despite extensive switching across the board
- The CERBERUS board uses the available space—as determined by the micro-ATX standard—to increase the distance between components and improve signal integrity
  - Separating the VGA Circuit from the Computer Proper can be helpful
  - The inner metal planes also provide EM insulation between top and bottom routing layers

Signal buffering, filtering and termination

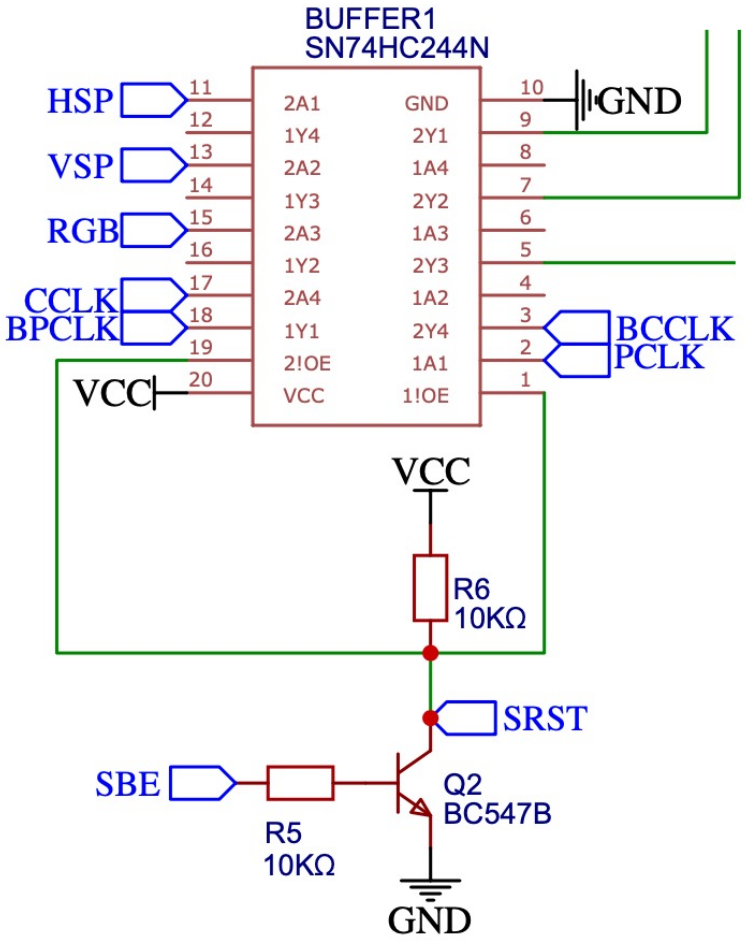
- The RGB signal produced by CERBERUS is double-buffered through a 74HC244 IC followed by a high-current transistor (Q1)
  - It should be safe to connect CERBERUS to any VGA monitor
- The resistor connecting the emitter of Q1 to GND is valued at 75 Ohms to match the impedance of the monitor
  - This avoids signal reflections
- The synch pulse signals are not double-buffered, but pass through 68 Ohms current-limiting resistors instead
  - They are also filtered by 470pF capacitors to reduce ringing
- All clock signals in CERBERUS are buffered through 74HC244 ICs
  - The CPU clocks are also filtered by 100pF capacitors to reduce ringing
- The data and addresses buses in CERBERUS are pulled *down* to ground by two resistor banks clearly marked on the board
  - For CMOS devices, as used in CERBERUS, logic outputs driving the buses high can easily overcome the pull-down resistors without much of a current drain
- All control signals in CERBERUS are pulled *high* (non-active) by pull-up resistors
  - CERBERUS has no control bus, but a control nexus (SPACER) instead
  - All control outputs from memories, CPUs and CAT feed into logic circuitry within SPACER
  - Therefore, these outputs must *always* be driven; if they were allowed to float, the logic in SPACER would produce unpredictable results

CPLD power-on reset

- The Microchip ATF CPLDs used in CERBERUS have a built-in power-on reset feature
  - Without this power-on reset, the internal counters won't start up correctly and the ICs won't work properly (often not even at all)
- But the feature only works if…
  - The Vcc signal rises *monotonically* and then remains stable
  - The CPLD is fully powered on *before* the first clock pulse arrives
- As it turns out, these conditions are not trivial to meet without dedicated electrical provisions
- Therefore, CERBERUS uses dedicated circuitry to *force* a correct power-on reset
  - The RST input to the CPLDs is active until the CPLDs are fully powered on
  - The clock input to the CPLDs is disabled until the ICs are fully powered on
- On the CERBERUS Schematics, this dedicated circuitry encompasses:
  - Transistors Q2 and Q3
  - Resistors R5, R6, R11 and R12
  - BUFFER1 and BUFFER2 (which are also used for other purposes)

Power-on reset circuitry

- Two pins of each ATF1508AS CPLD (SCUNK and SPACER) are dedicated to:
  - A global ‘RST’ (ReSeT) input that clears every flip-flop in the chip
  - A ‘BE’ (Buffer Enable) output that goes high as soon as the chip is fully powered on
- The ‘BE’ output is inverted by a transistor and then used as chip enable signal for a 74HC244 buffer, through which the CPLD’s clock is routed
  - Therefore, while ‘BE’ is low—during power on—the 74HC244 is disabled and no clock gets to the CPLD
  - The clock is only enabled after power on is complete and ‘BE’ goes high
- The inverted ‘BE’ output also serves as global reset for the same CPLD
  - While ‘BE’ is low—i.e. during power on—the reset is active and clearing the CPLD’s internal flip-flops
- CAVIA does not need its own power-on reset circuitry, as it is reset by SCUNK

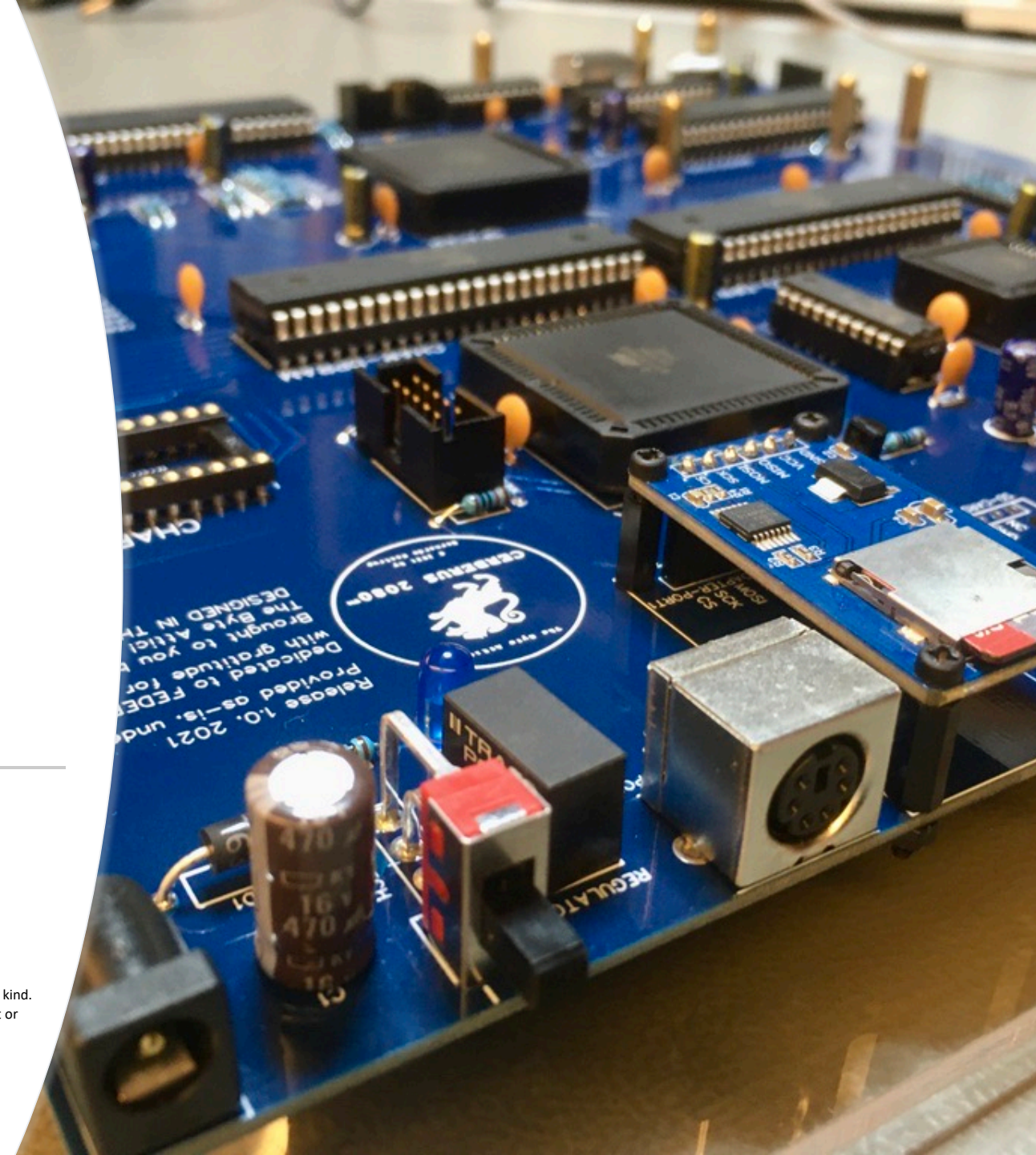


The Byte Attic's

CERBERUS 2080™

© 2021 by
Bernardo Kastrup

Custom IC design notes



Custom IC design

- These brief notes aim to merely *complement* the extensive commentary embedded in the .PLD files provided with CERBERUS 2080's distribution, which contain the full hardware description of the three custom ICs in CERBERUS: SCUNK.PLD, CAVIA.PLD and SPACER.PLD
- The CUPL hardware design language used is extensively described in the CUPL documentation also provided with CERBERUS 2080's distribution
- The CUPL software is freely-available and can also be found in CERBERUS 2080's distribution, so you can experiment with changing the design of SCUNK, CAVIA or SPACER, if you so dare

- Very briefly, here is the CUPL syntax:
  - '`&`' means a logical AND
  - '`#`' means a logical OR
  - '`!`' means a logical INVERSION
  - '`SIGNAL.t`' is the input to a toggle flip-flop whose output is `SIGNAL`
  - '`SIGNAL.d`' is the input to a D flip-flop whose output is `SIGNAL`
  - '`SIGNAL.ck`' is the clock input to a flip-flop whose output is `SIGNAL`
  - '`SIGNAL.ce`' is the clock enable input to a flip-flop whose output is `SIGNAL`
  - '`SIGNAL.ar`' is the asynchronous reset input to a flip-flop whose output is `SIGNAL`