Mit Lötkolben, Wire-Wrap-Pistole und Assembler

Z80 Selbstbaurechner

Prof. Dr. Bernd Ulmann

März 2014

Einleitung

Warum sollte man heute, im 21. Jahrhundert, über einen 8 Bit-Prozessor wie den Z80 sprechen oder gar rund um diesen Prozessor kleine Computersysteme aufbauen? Gründe dafür gibt es einige:

- Es macht einfach Spass – viel mehr, als z.B. mit einem bereits fertigen Raspberry Pi-Board zu experimentieren.
- Es liefert einen sehr guten Einstieg in die Digitalelektronik mit im wahrsten Sinne des Wortes anschaulichen Bauelementen.
- Die Softwareseite eines solchen Unterfangens ist hinreichend einfach, dass es keiner hochkomplexen IDE und Toolchain bedarf, um Programme, einen einfachen Monitor oder sogar ein (ganz) kleines Betriebssystem zu schreiben.

Der „akademische Nährwert“ einer solchen Spielerei ist nicht zu unterschätzen:

- Moderne Prozessoren sind einer verständlichen Darstellung im Rahmen einführender Veranstaltungen wie „IT-Basics“ oder „Betriebssysteme Theorie“ allein aufgrund ihrer Komplexität nicht zugänglich.
- Moderne Software ist ebenfalls viel zu komplex, um im Rahmen einer solchen Veranstaltung behandelt zu werden, so dass letztlich die Frage danach, wie z.B. ein Betriebssystem wirklich funktioniert, etwas unklar bleiben muss.

Aus diesem Grund (und zugegebenermassen auch aus Freude am Basteln :-)) wurde ein kleiner Einplatinenrechner auf Basis des Z80-Prozessors entwickelt, für den auch ein einfacher Monitor in Assembler geschrieben wurde.

Wire-Wrap-Implementation

Wenn man Wire-Wrap-Werkzeug und Erfahrung im Umgang damit hat, lassen sich damit extrem schnell recht komplexe Schaltungen aufbauen. Entsprechend wurde das erste Z80-System, das im Folgenden beschrieben wird, auf zunächst einer, später zwei Europa-Lochrasterplatinen (Maße 100 mm$\times$ 160 mm) aufgebaut.

Beim Wire-Wrappen wird mit Hilfe einer Wire-Wrap-Pistole ein feiner Draht fest um viereckige Stifte herumgewickelt, wodurch dieser an den Stiftkanten mit jedem verschweisst. Diese Verbindungen sind bezüglich ihrer Zuverlässigkeit traditionellen Lötverbindungen sogar überlegen.

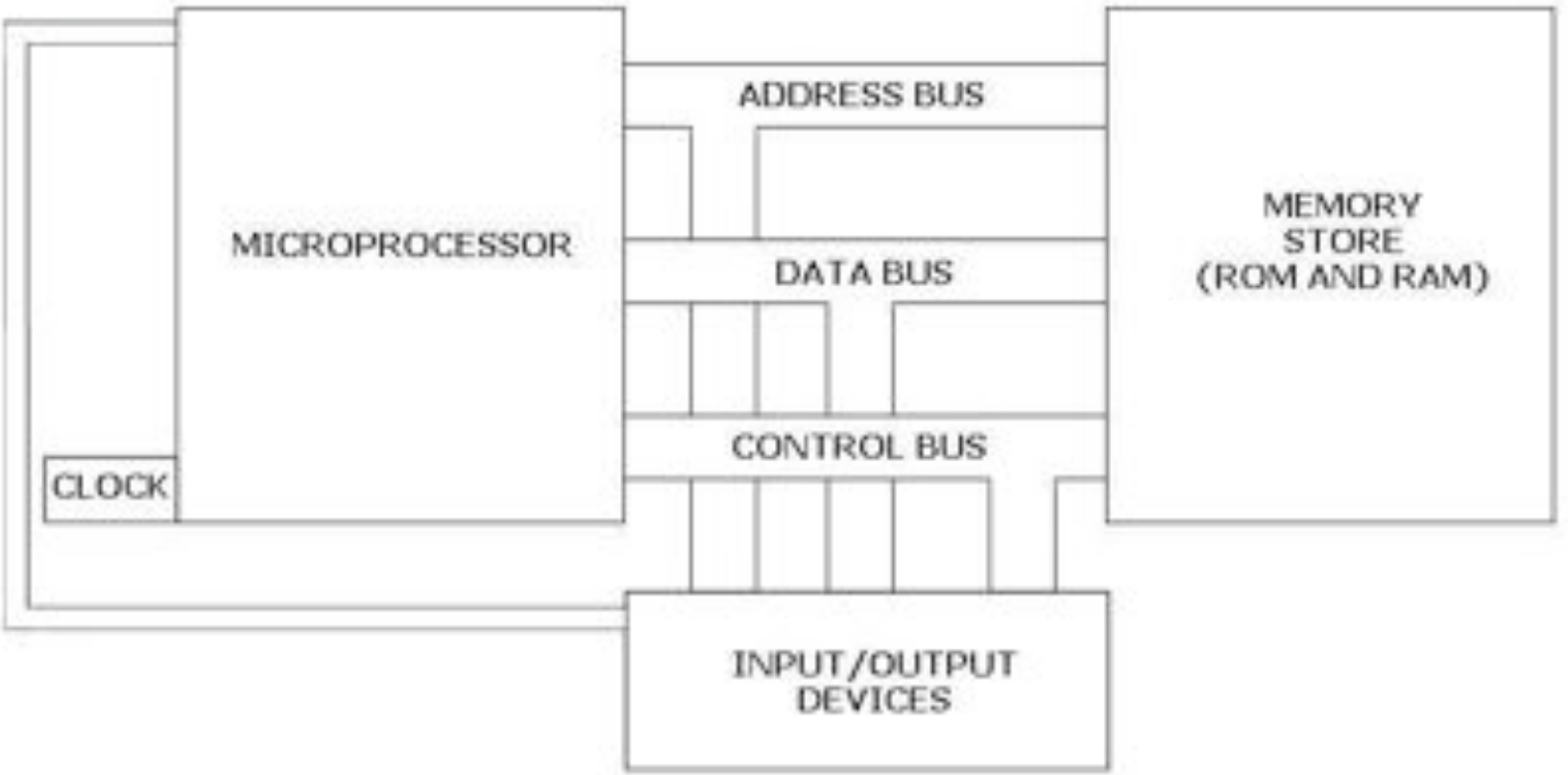
Eigenschaften

Dieser kleine Beispielrechner verfügt über die folgenden Eigenschaften:

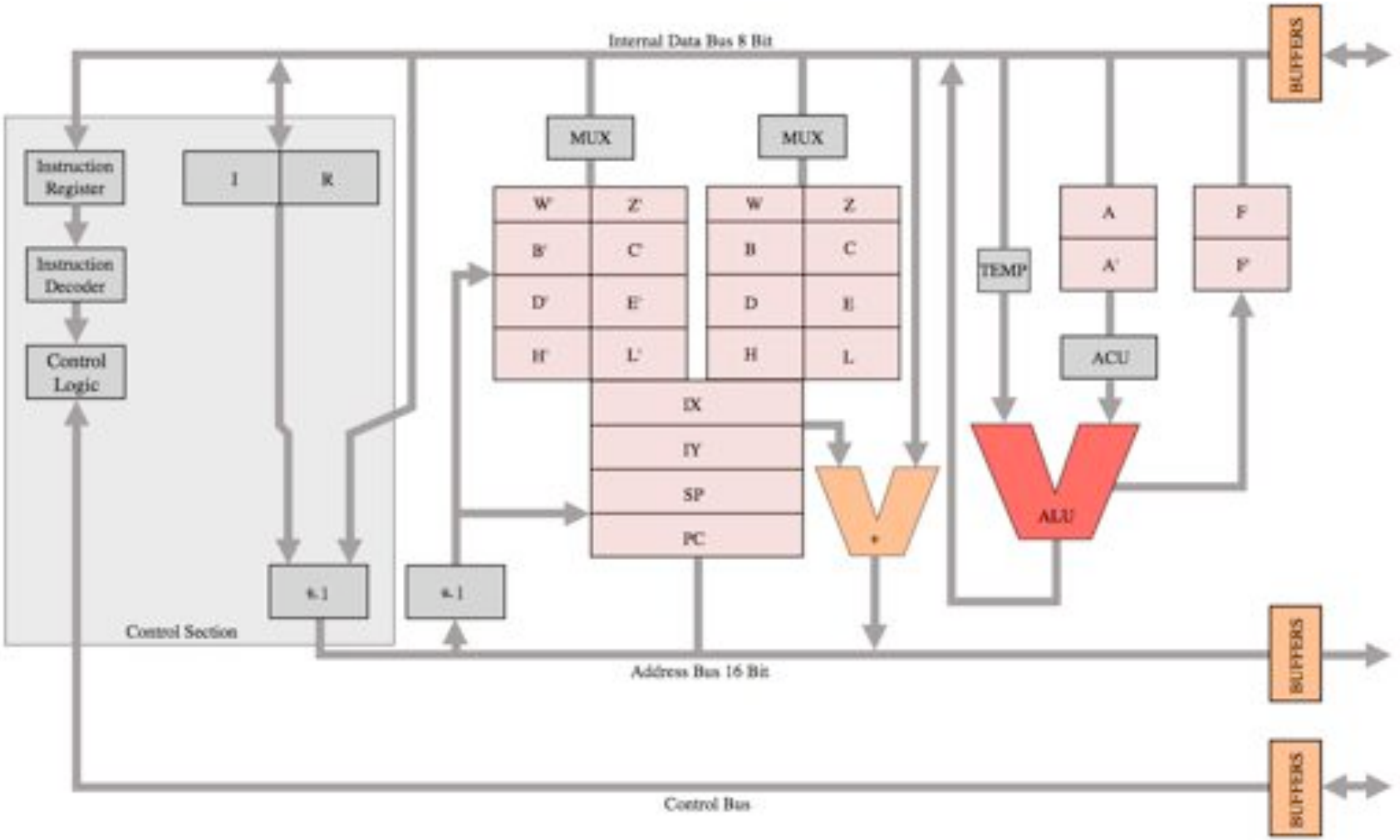
- Eine Karte für die CPU, eine für einen einfachen IDE-Controller.
- 64 kB Adressraum, davon 32 kB EPROM ($0000 bis $7FFF) für ein kleines Betriebssystem, und 32 kB RAM ($8000 bis $FFFF).
- Anbindung an die Aussenwelt über eine serielle Schnittstelle.
- Nur eine Versorgungsspannung von +5 Volt bei wenigen 100 mA nötig.
- Einfaches Bussystem auf Basis 64-poliger VG-Leisten.
- Nicht unterstützt wird gegenwärtig DMA-Betrieb.

Die folgenden Bilder zeigen die Entstehung des Rechners:

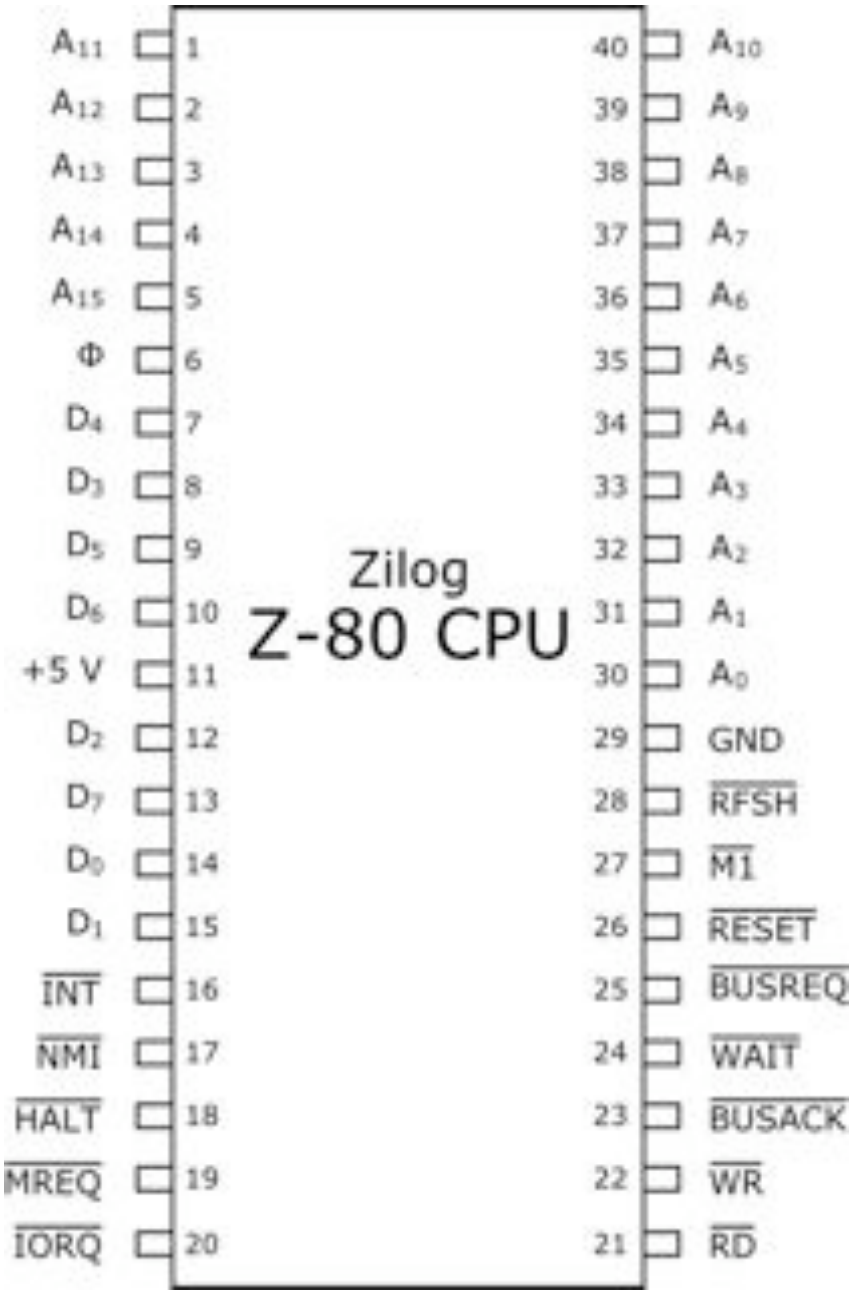
CPU-Platine



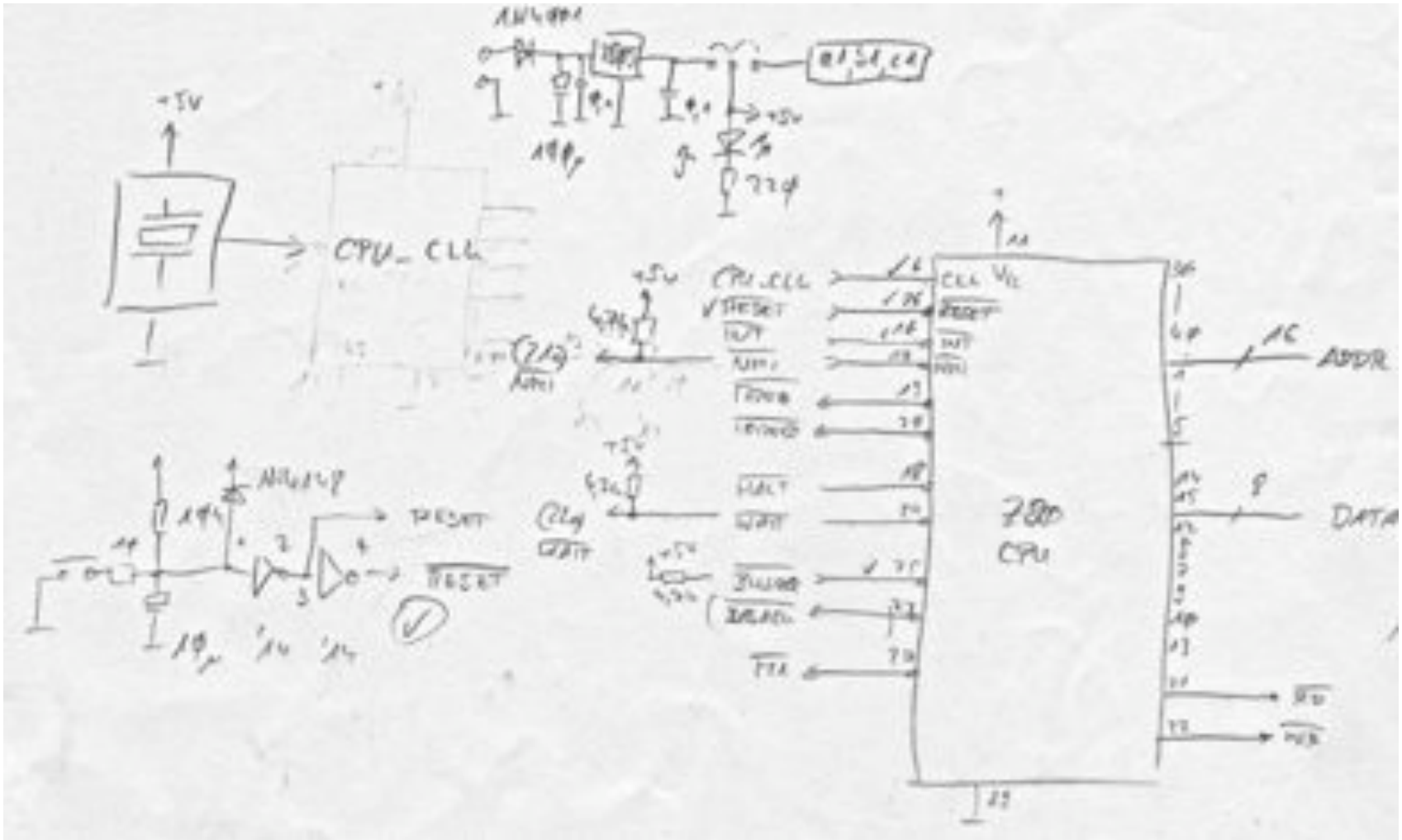
Quelle: `http://www.arcadegeek.co.uk/how-do-games-work`, Stand 18.02.2014.



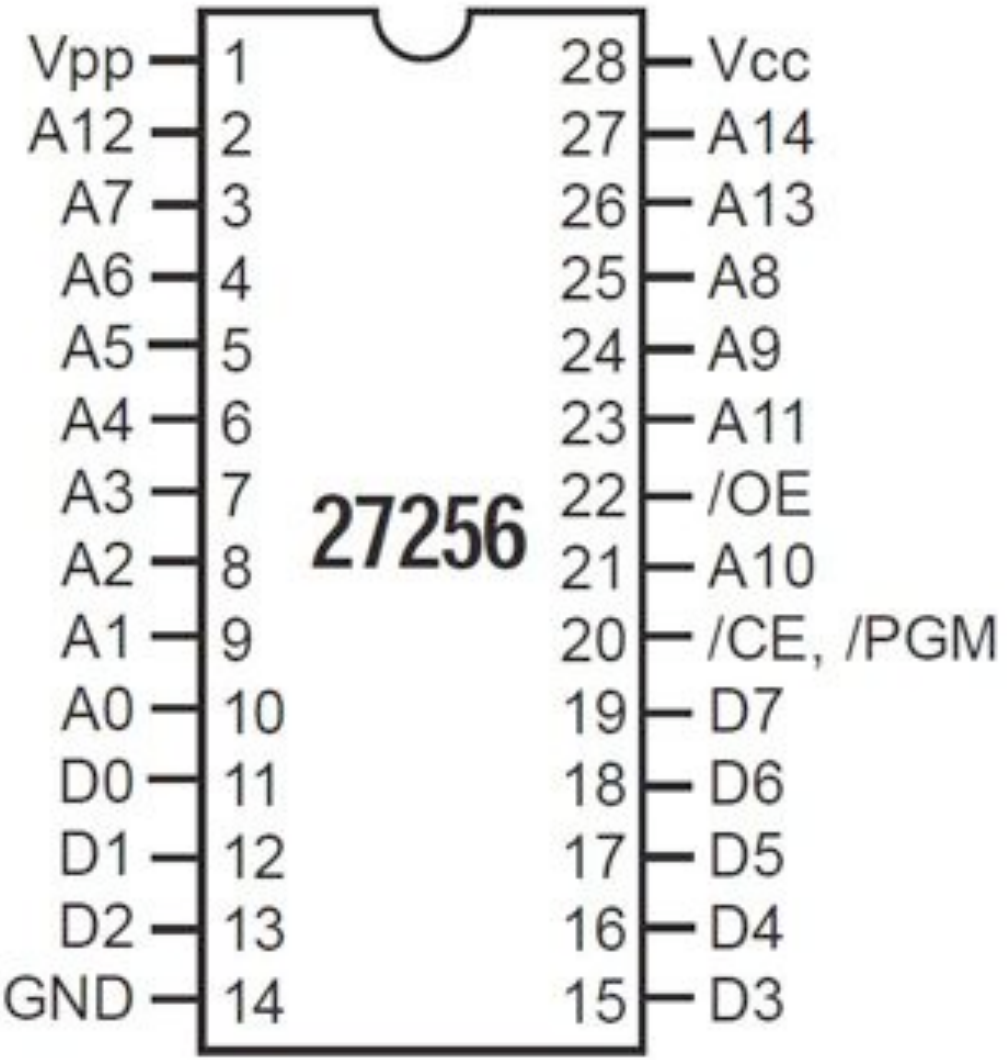
Quelle: `http://upload.wikimedia.org/wikipedia/commons/d/db/Z80_arch.svg`, Stand 28.02.2014

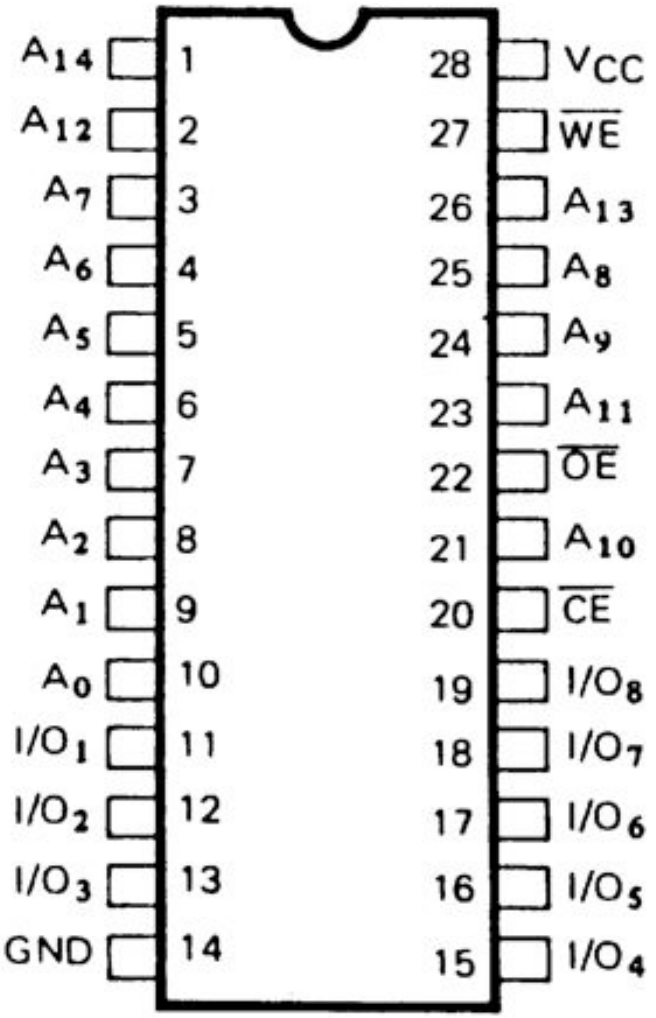

Zilog
Z-80 CPU
A11 1
A12 2
A13 3
A14 4
A15 5
Φ 6
D4 7
D3 8
D5 9
D6 10
+5 V 11
D2 12
D7 13
D0 14
D1 15
INT 16
NMI 17
HALT 18
MREQ 19
IORQ 20
40 A10
39 A9
38 A8
37 A7
36 A6
35 A5
34 A4
33 A3
32 A2
31 A1
30 A0
29 GND
28 RFSH
27 M1
26 RESET
25 BUSREQ
24 WAIT
23 BUSACK
22 WR
21 RD

Rund um die CPU

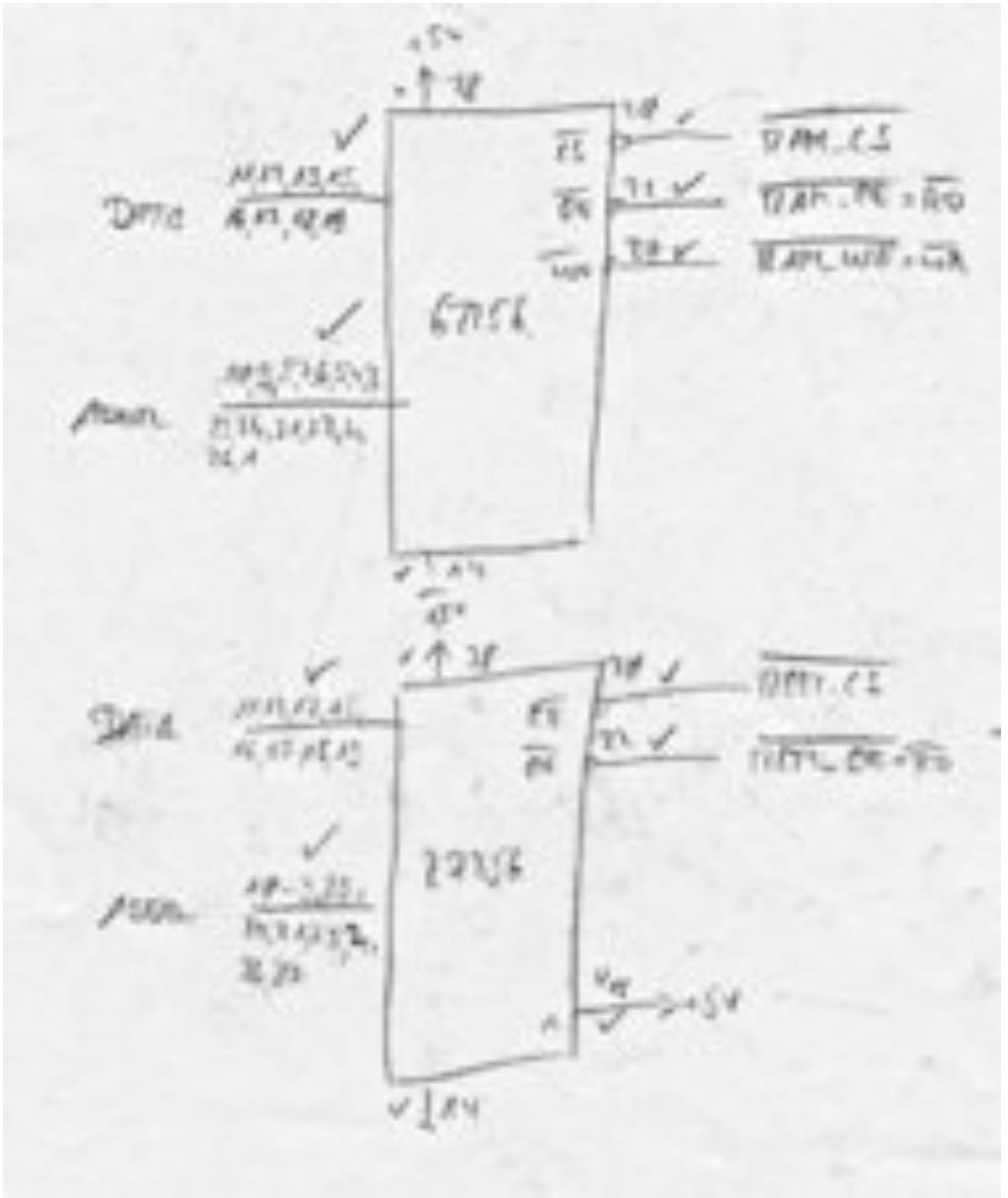


Das EPROM


27256
Vpp 1
A12 2
A7 3
A6 4
A5 5
A4 6
A3 7
A2 8
A1 9
A0 10
D0 11
D1 12
D2 13
GND 14
28 Vcc
27 A14
26 A13
25 A8
24 A9
23 A11
22 /OE
21 A10
20 /CE, /PGM
19 D7
18 D6
17 D5
16 D4
15 D3

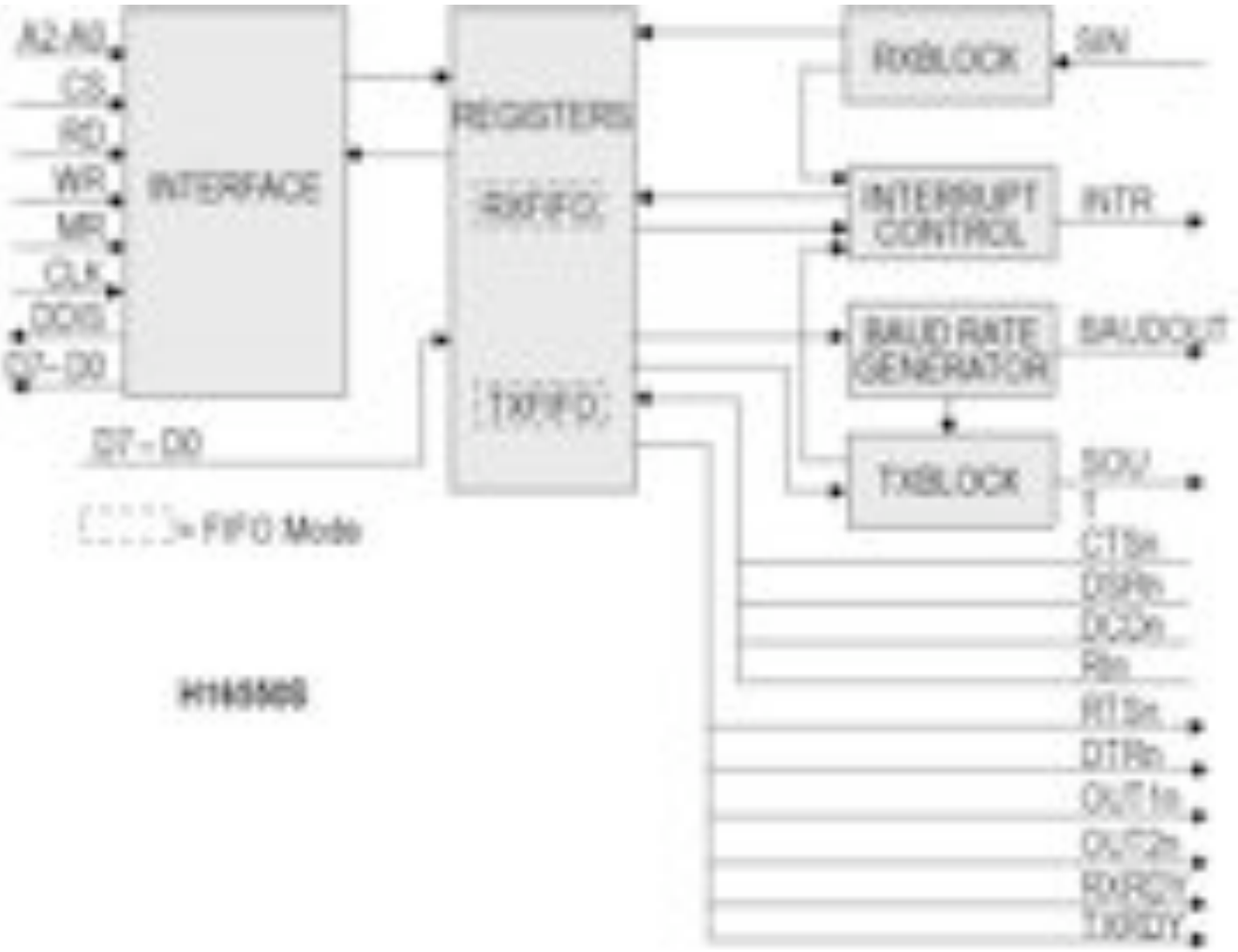

A14 1
A12 2
A7 3
A6 4
A5 5
A4 6
A3 7
A2 8
A1 9
A0 10
I/O1 11
I/O2 12
I/O3 13
GND 14
28 VCC
27 WE
26 A13
25 A8
24 A9
23 A11
22 OE
21 A10
20 CE
19 I/O8
18 I/O7
17 I/O6
16 I/O5
15 I/O4

Rund um RAM und EPROM

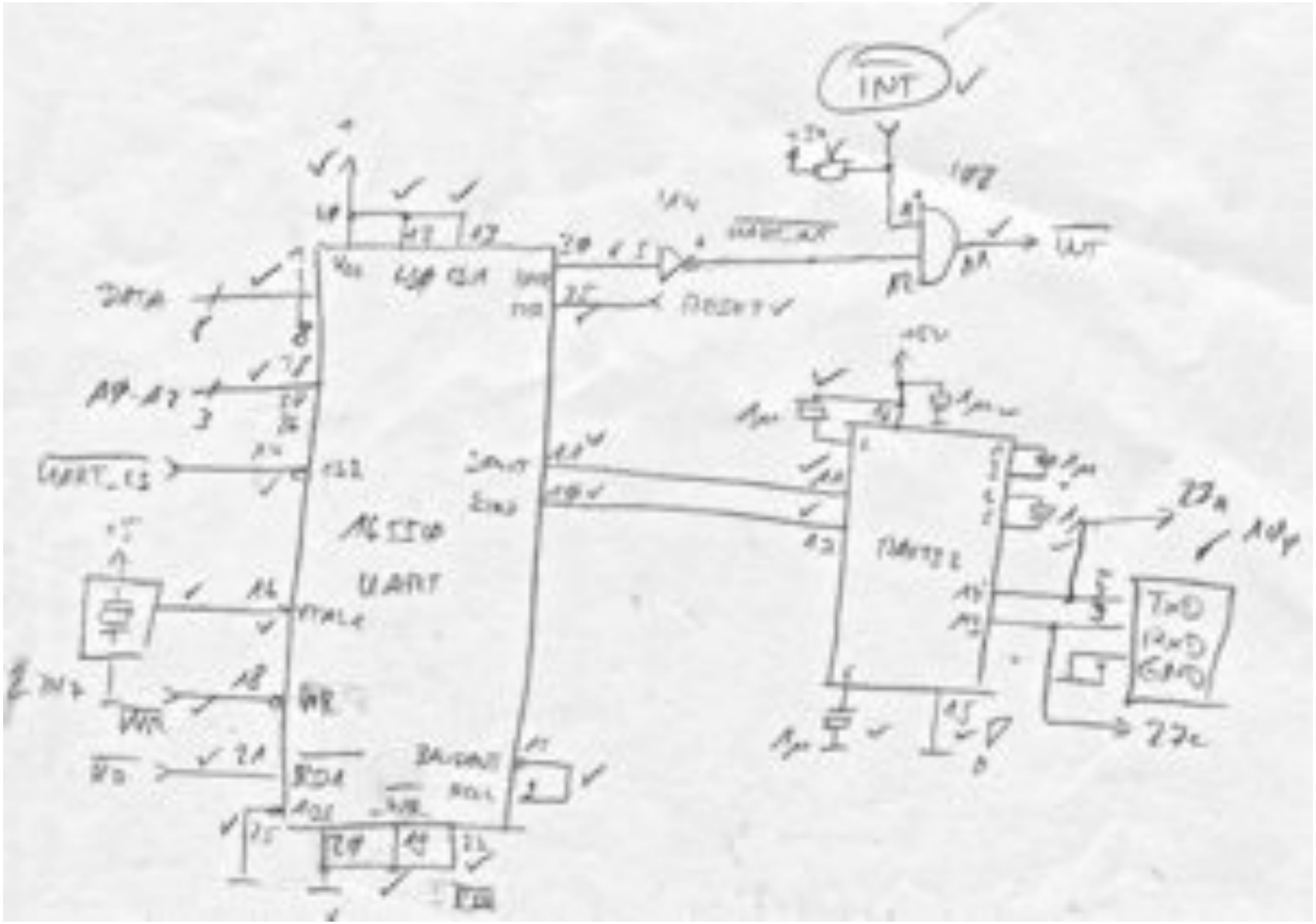


Eines fehlt noch...

. . . nämlich eine serielle Schnittstelle. . .



Quelle: `http://www.cast-inc.com/ip-cores/uarts/h16550s/`

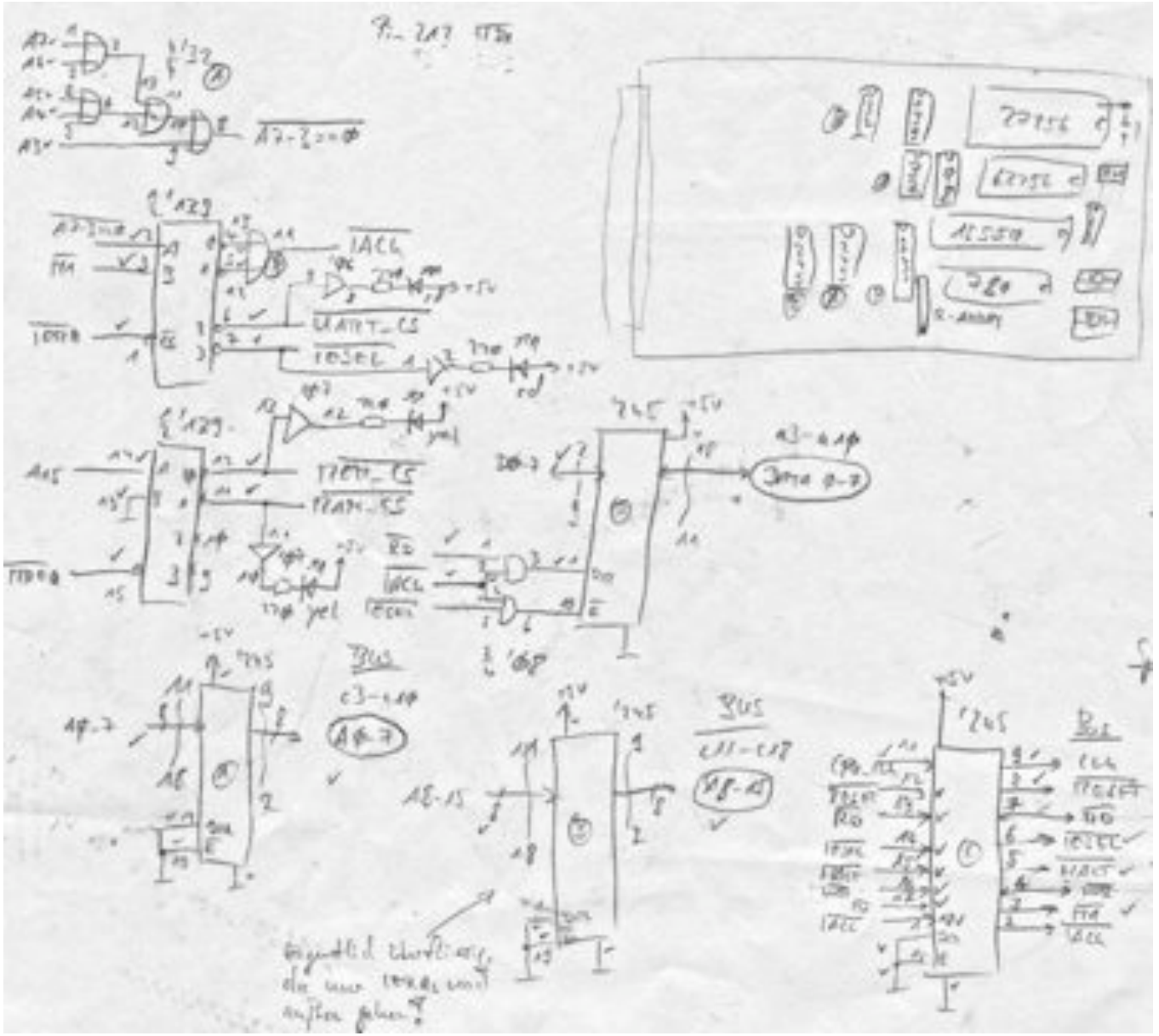

INT
UART
TxD
RxD
GND

Nun fehlt aber noch etwas Essenzielles: Eine Adressdekodierung, die steuert, welcher Baustein wann, d.h. unter welcher sogenannten *Adresse* angesprochen wird.

Der Z80-Prozessor ist diesbezüglich ausgesprochen angenehm, da er spezielle IO-Instruktionen sowie entsprechende Steuersignale anbietet, mit denen unterschieden werden kann, wann auf Speicher und wann auf IO-Geräte zugegriffen wird.

Bei der Gelegenheit können gleich noch ein paar *Bustreiber* vorgesehen werden, um das System später erweitern zu können.

Adressdekodierung



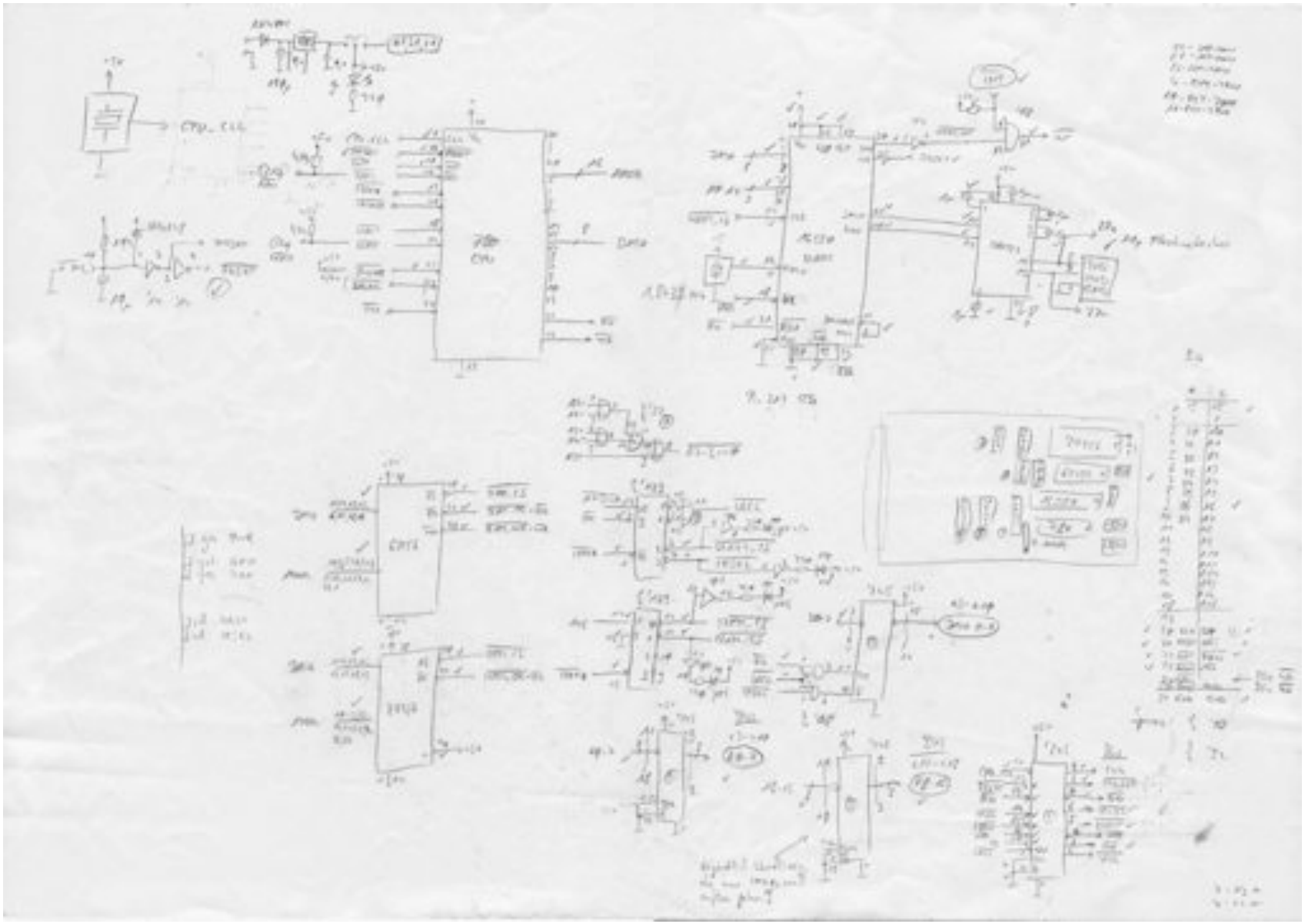
Alles zusammen...

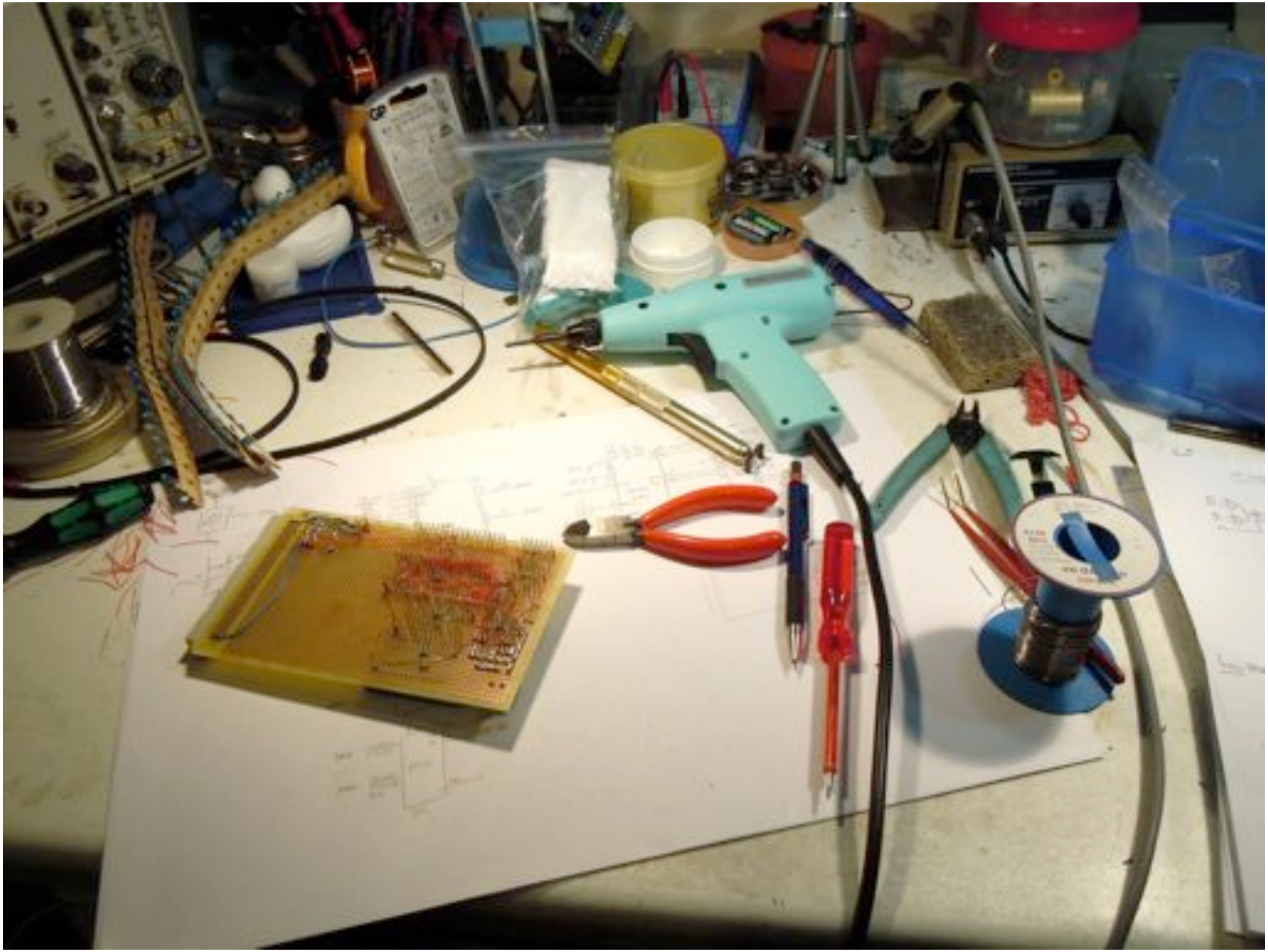
Fasst man diese Grundelemente,

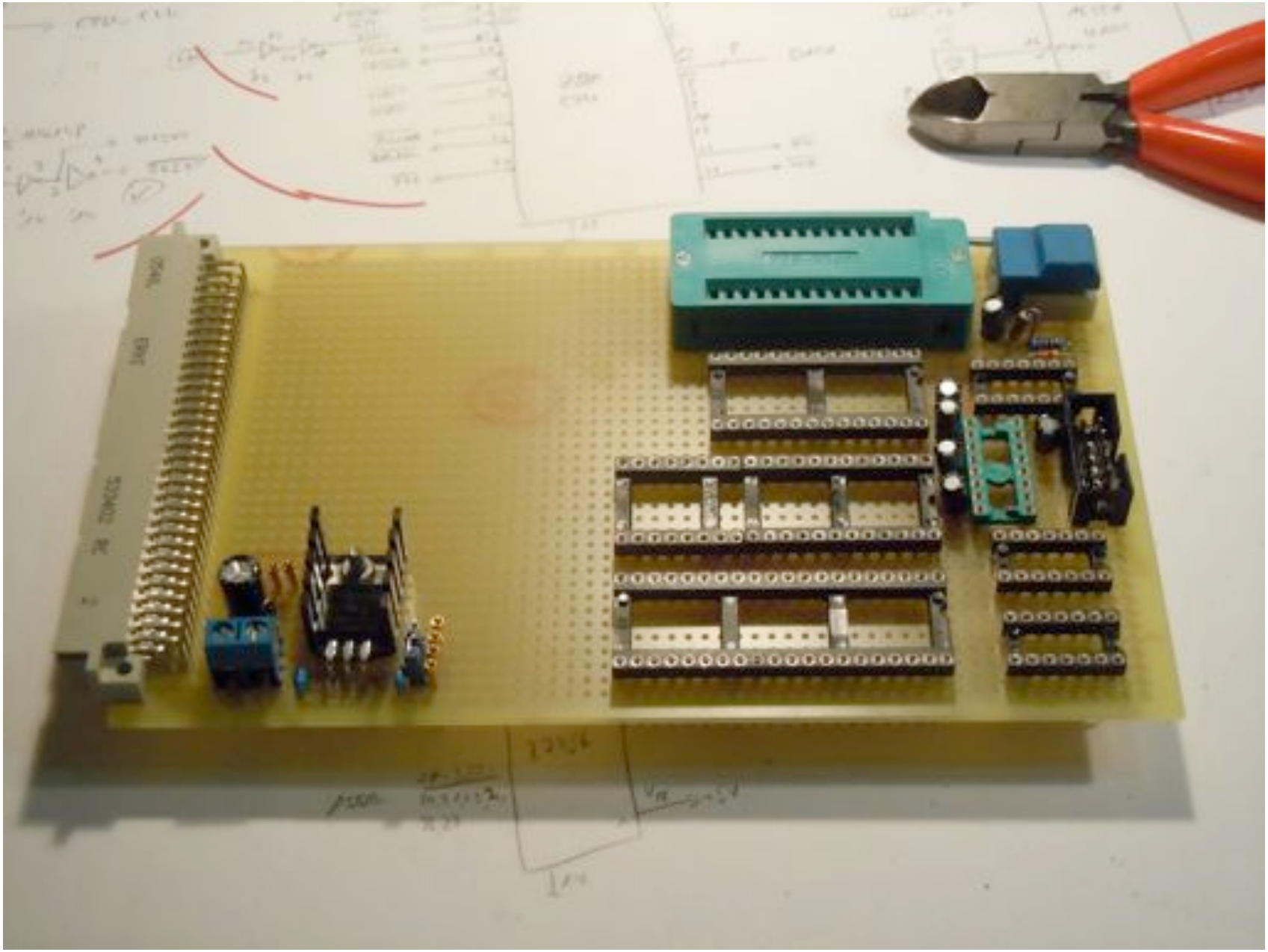
- CPU, Takt- und Resetgenerator, Stromversorgung
- RAM und EPROM
- Serielle Schnittstelle
- Adressdekoder und Bustreiber

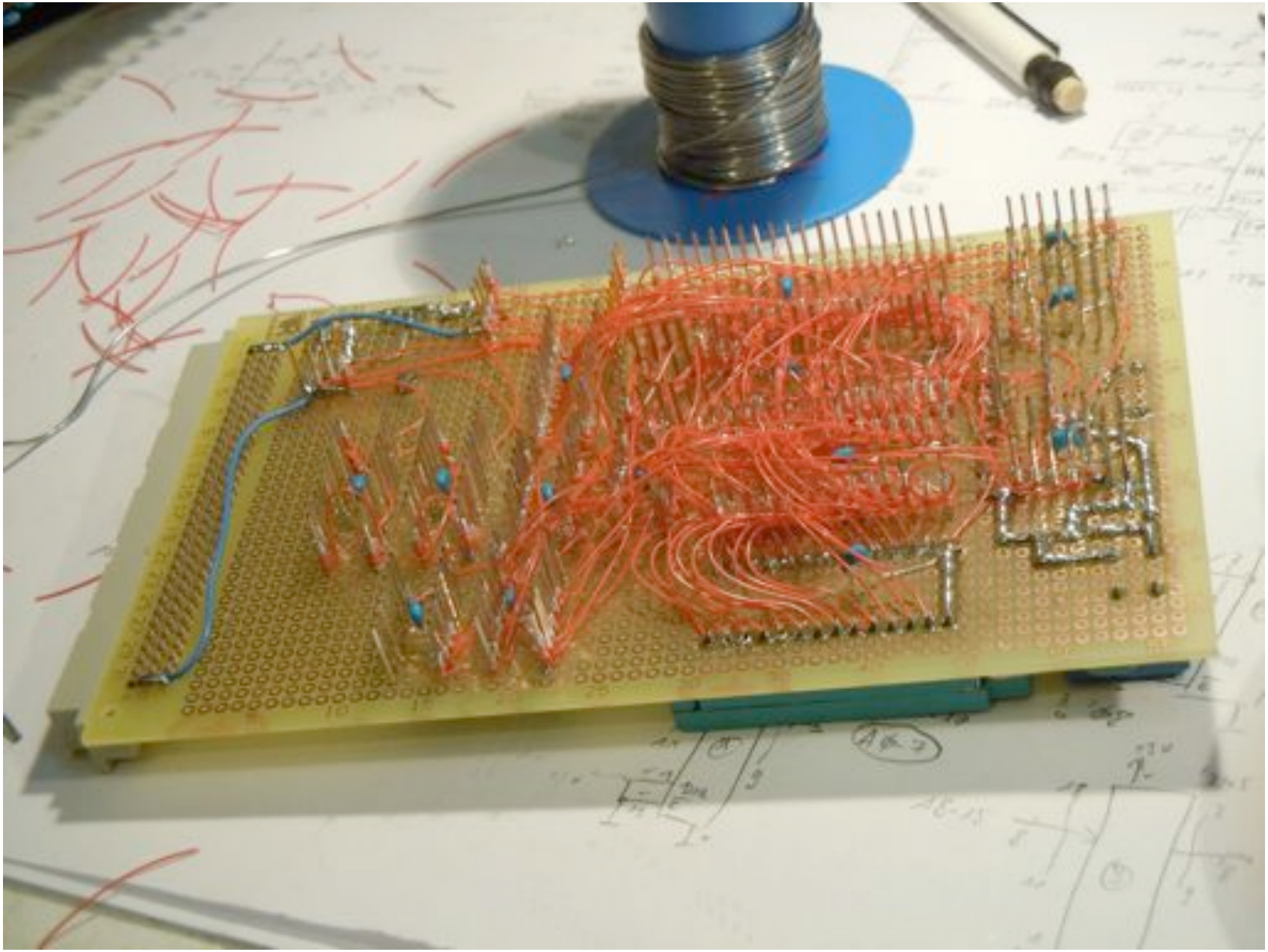
zusammen, ergibt sich Folgendes:

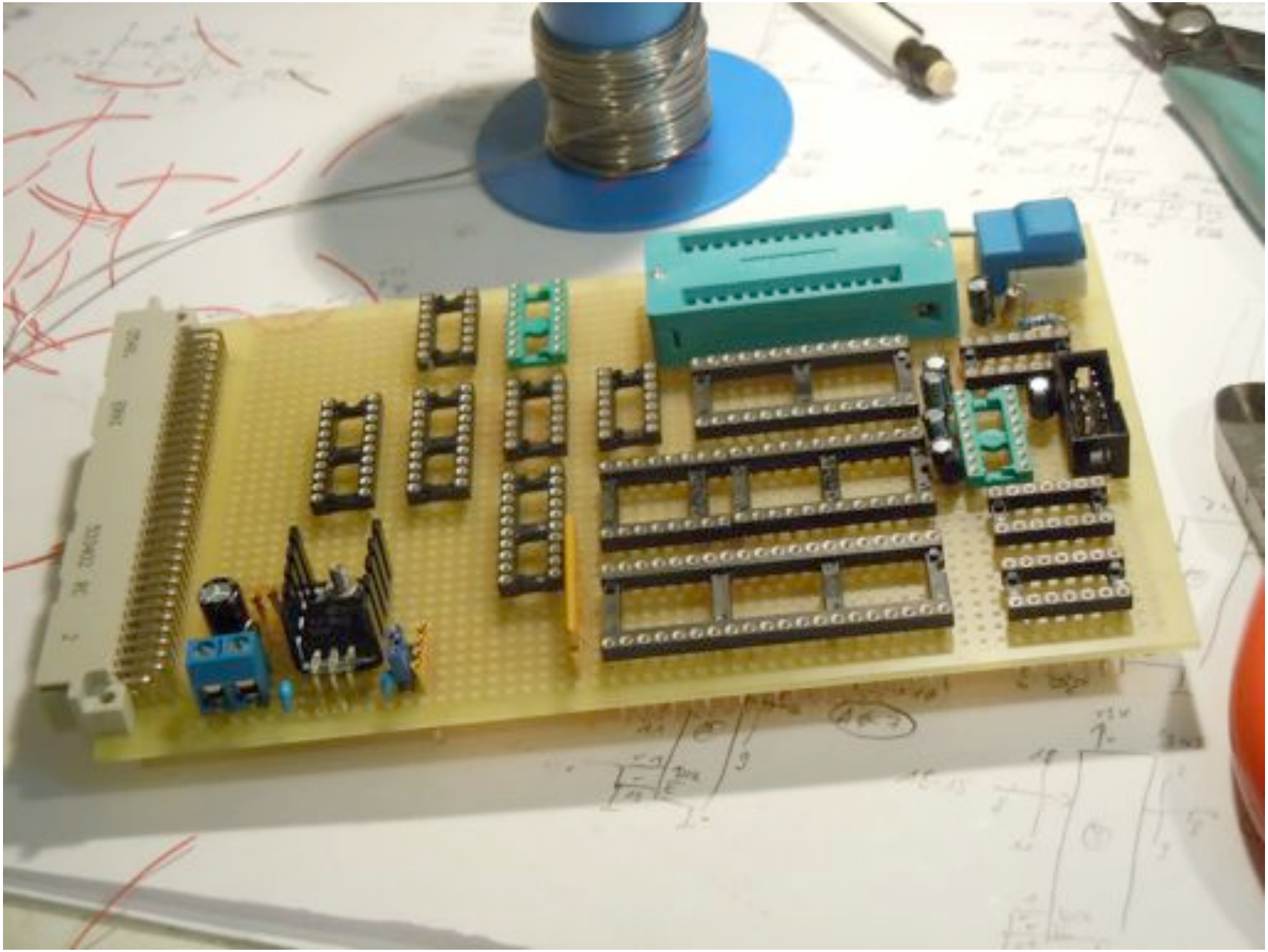
Gesamtschaltplan

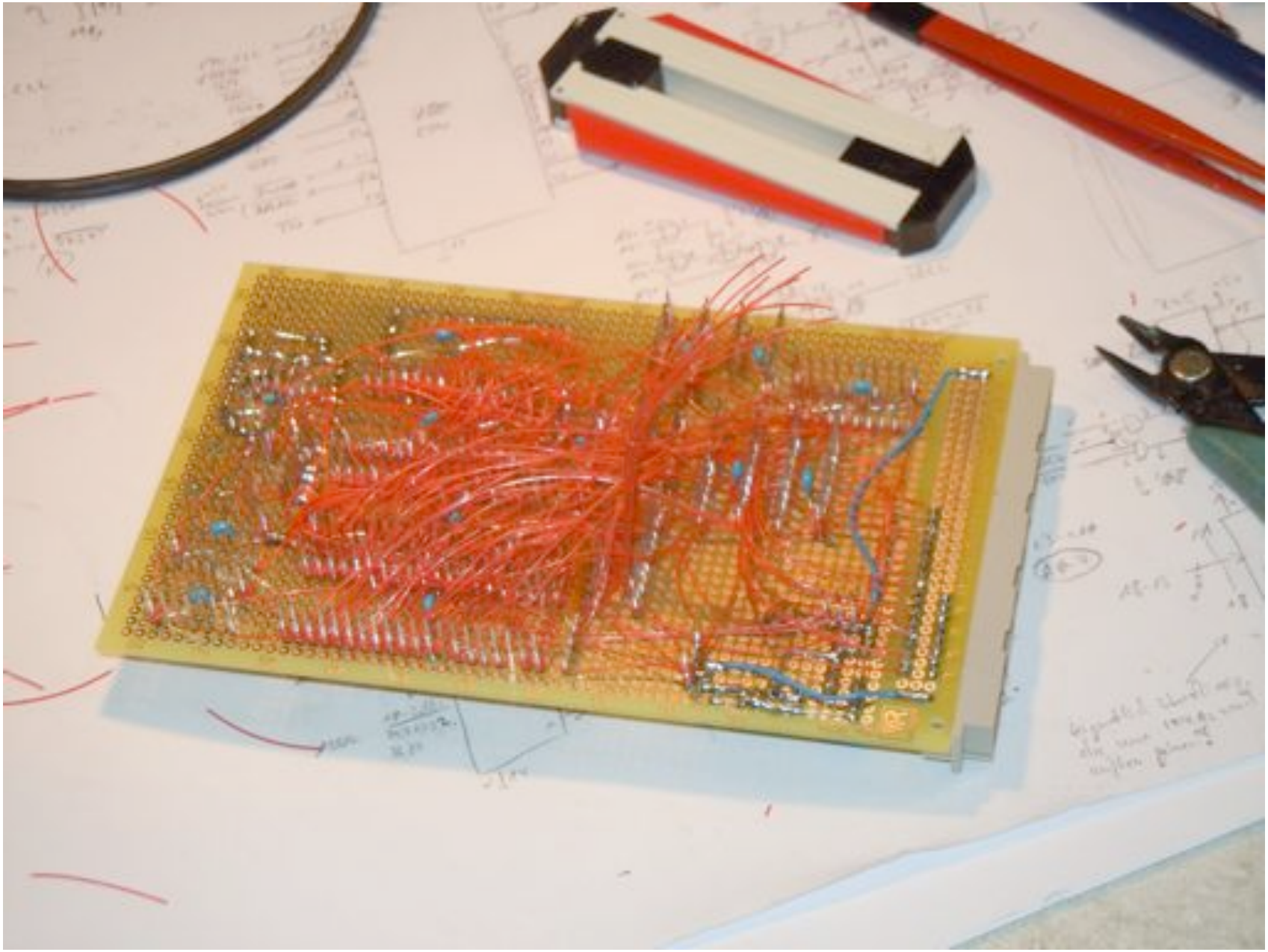


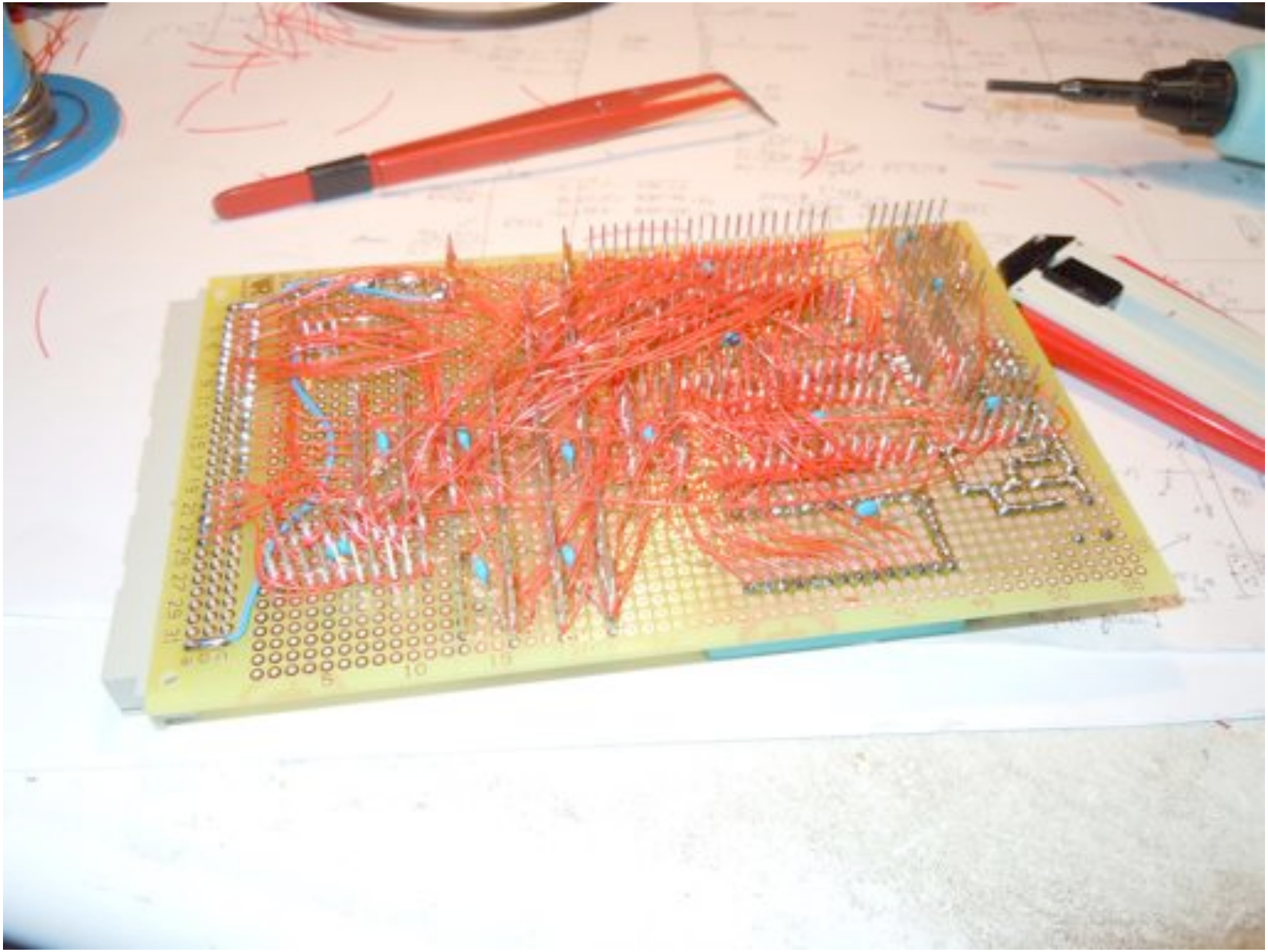






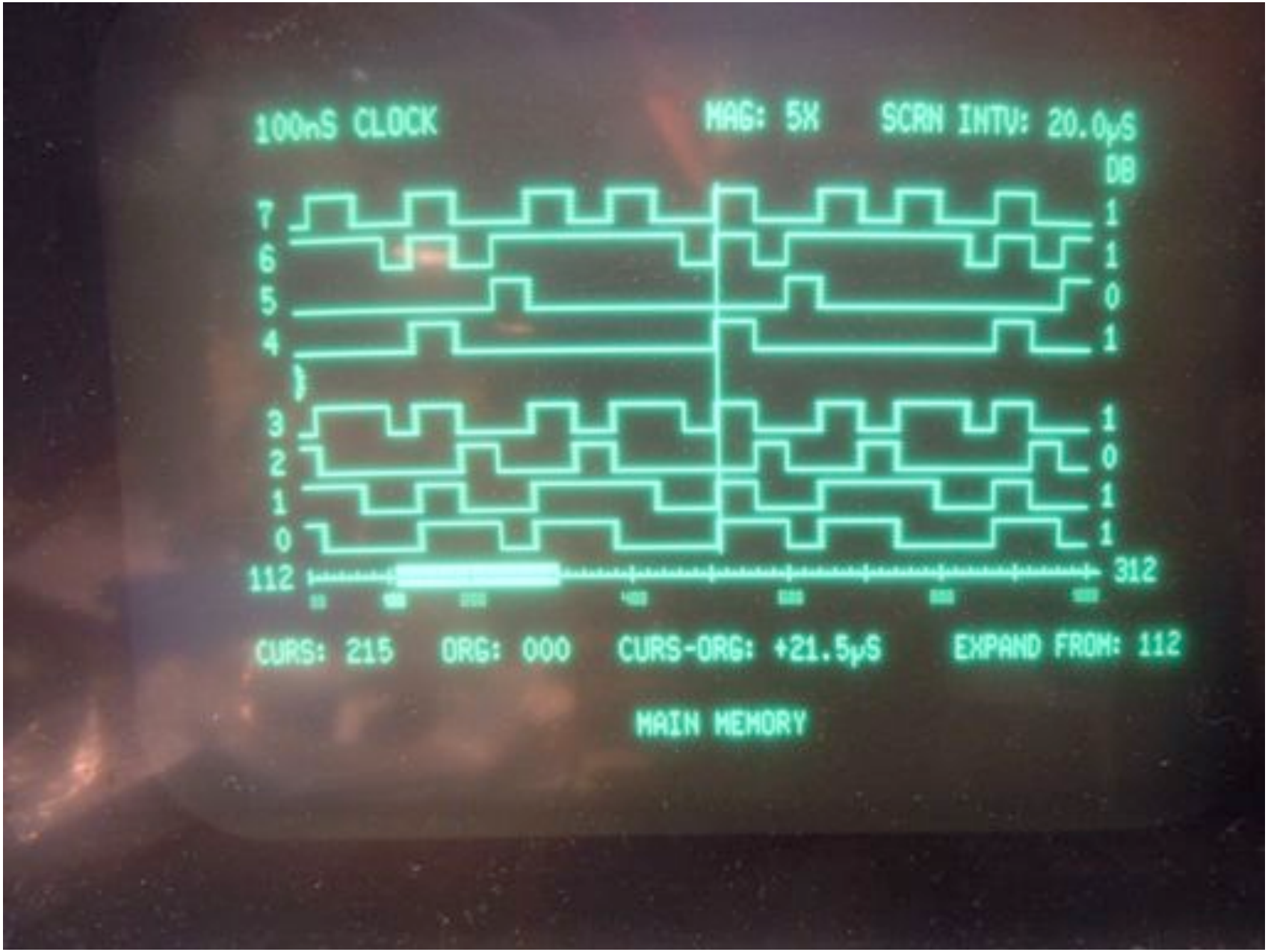






Erste Inbetriebnahme

Erste Inbetriebnahme



Erste Inbetriebnahme

Software

- Um nun mit dieser Karte etwas anfangen zu können, benötigt man ein kleines Betriebssystem, einen *Monitor*.
- Nur, worauf entwickelt man den?
- Das ist heute zum Glück viel einfacher als zur Schulzeit des Autors, als man EPROMs noch wirklich per Hand mit einer einfachen Schaltung und vielen Schaltern mühevoll programmiert und Assemblerprogramme per Hand in Maschinencode übersetzt hat:
- Man verwendet einen Cross-Assembler. Im vorliegenden Fall wurde `zasm`[1], der klaglos auf UNIX-Plattformen, in diesem Fall Mac OS X, läuft.
- Was benötigt man noch? Einen EPROM-Programmierer und ein EPROM-Löschgerät:

[1] Siehe `http://k1.dyndns.org/Develop/projects/zasm/distributions/`, Stand 08.06.2013.


WARNING
MOS DEVICES ENCLOSED
L5000

Nun möchte man so etwas

```
eos     defb    0
        ld      hl, hello
        call    puts
        ...
hello   defb    "Hello world!", eos
```

oder so etwas

```
loop    call    getc
        call    putc
        jr      echo
```

machen können, wofür man schon einige einfache Routinen benötigt:

```
puts            push    af
                push    hl
puts_loop       ld      a, (hl)
                cp      eos             ; End of string reached?
                jr      z, puts_end     ; Yes
                call    putc            ; Write character
                inc     hl              ; Increment character pointer
                jr      puts_loop       ; Transmit next character
puts_end        pop     hl
                pop     af
                ret
putc            call    tx_ready
                out     (uart0), a
                ret
tx_ready        push    af
tx_ready_loop   in      a, (uart5)
                bit     5, a
                jr      z, tx_ready_loop
                pop af
                ret
```

Wenn das funktioniert, benötigt man noch ein paar Routinen, um Daten von der seriellen Schnittstelle einzulesen, grundlegende Stringoperationen durchzuführen etc. (Auszug):

```
getc            call    rx_ready
                in      a, (uart0)
                ret
rx_ready        push    af
rx_ready_loop   in  a, (uart5)
                bit 0, a
                jr  z, rx_ready_loop
                pop af
                ret
```

Dazu jedoch später mehr. . .

Massenspeicher

Ab einer gewissen Leistungsfähigkeit des Betriebssystems wünscht man sich irgendeine Form eines Massenspeichers. Was bietet sich an?

Kassettenlaufwerk: Verhältnismäßig einfach zu realisieren – entweder komplett in Software oder mit Hilfe eines UART-Bausteines mit FM-Modulator und -Demodulator. Nachteil: Langsam

Diskettenlaufwerk: Recht schnell, allerdings ist die Entwicklung eines Diskettencontrollers ausgesprochen eklig – der Autor hat das zuletzt im Alter von 16 Jahren gemacht und ringt immer noch mit dem traumatischen Erlebnis, eine PLL im elterlichen Backofen einem Burn-In unterziehen zu müssen. Das ist also auch keine Alternative.

IDE-Festplatte: IDE-Platten sind aus Hardwaresicht herrlich leicht zu interfacen, in Form von CF-Karten sind sie unschlagbar billig und schnell! Das ist die Lösung!

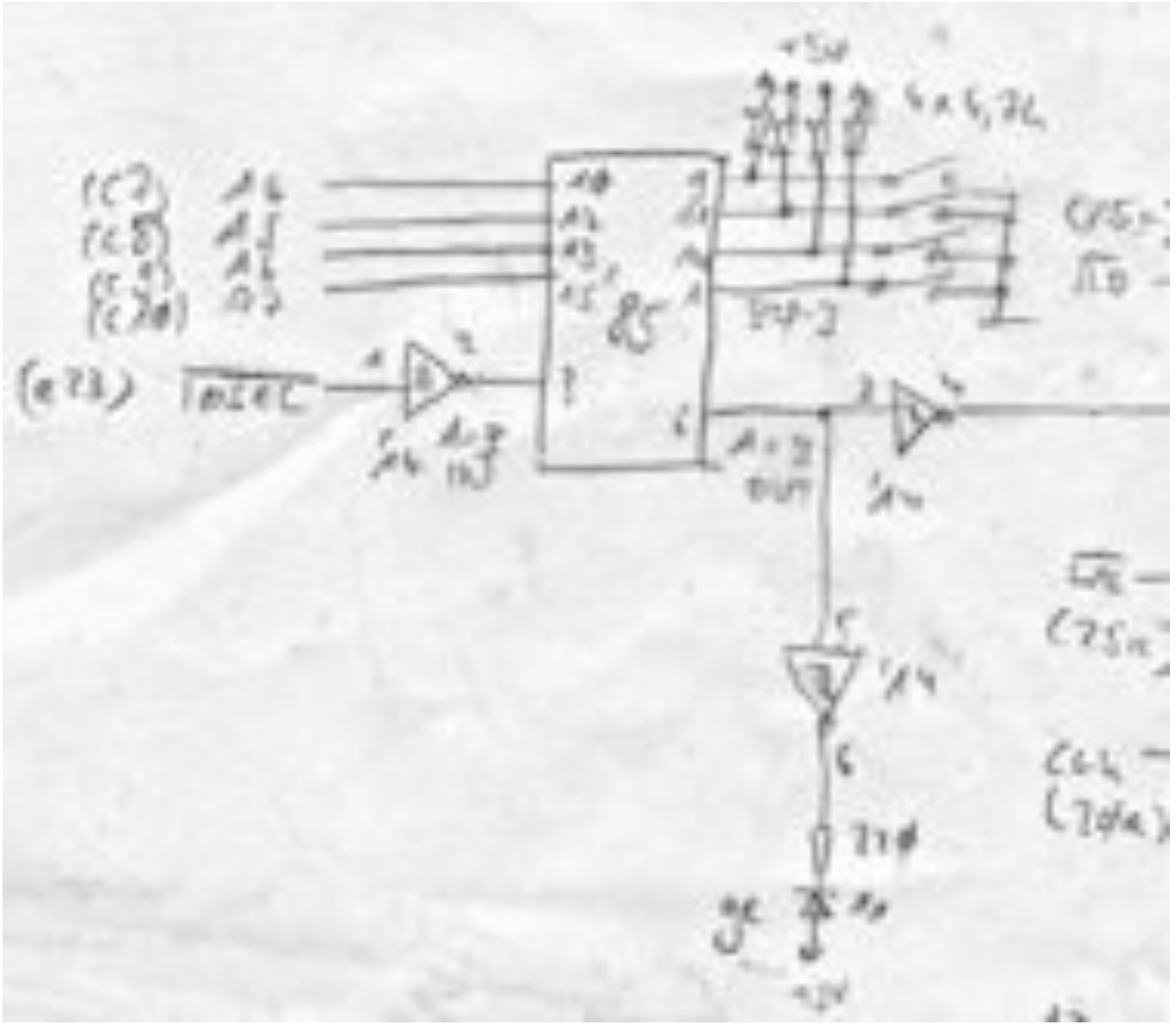
IDE-Controller

Also ein IDE-Controller... Im Prinzip trivial, da die IDE-Schnittstelle eigentlich schon ein Businterface ist, aber...

IDE ist ein 16 Bit-Bus, während der Z80 einen 8 Bit-Bus besitzt, man braucht also doch wieder etwas Elektronik... Die folgende Schaltung basiert auf einer Schaltung von `Phil` von Retroleum (`http://www.retroleum.co.uk/electronics-articles/an-8-bit-ide-interface/`).

Erst einmal ein wenig Adressdekodierung (schon wieder), weil das IDE-Interface eine Erweiterungskarte werden soll, die an einem Bus steckt:

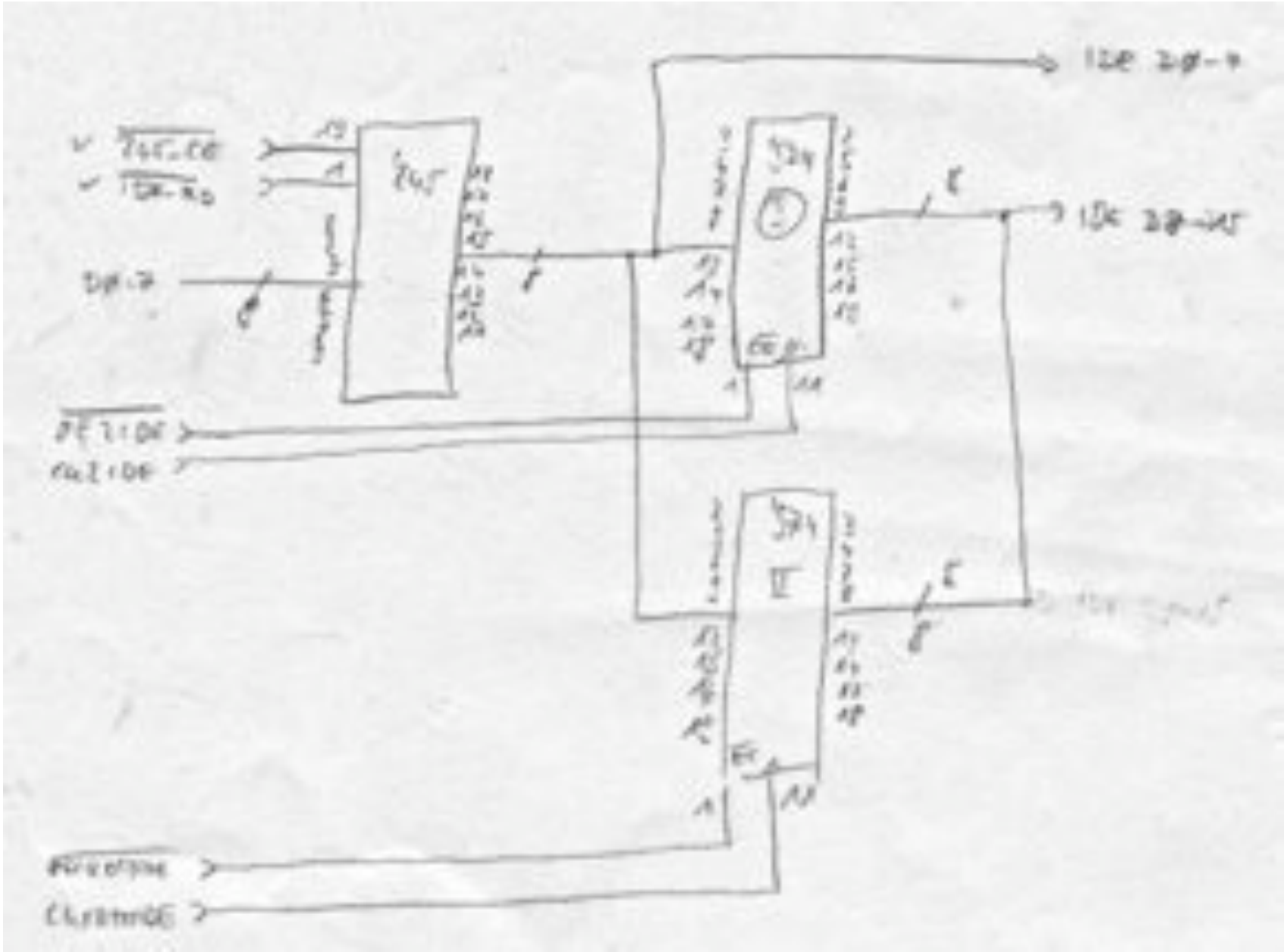
Adress-Dekodierung



Nun bleibt noch das Problem, aus einem 8 Bit-Datenbus einen 16 Bit-Bus zu machen und umgekehrt...

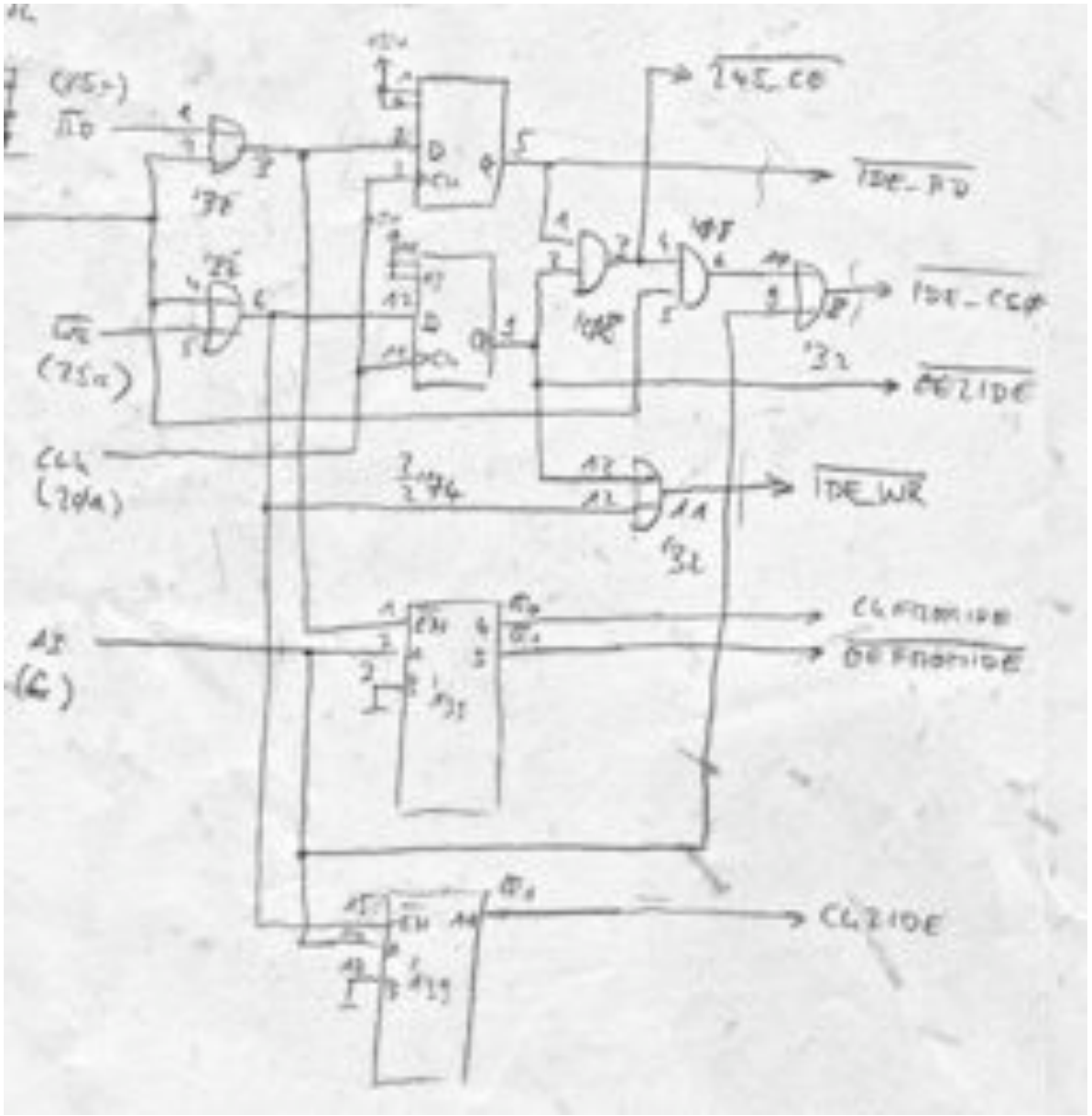
Die Idee ist hierbei, einen *Bustreiber* für die unteren 8 Bit zu verwenden und die oberen 8 Bit in je einem *Register*, eines für lesenden und eines für schreibenden Zugriff, zu speichern:

Datenbuskonverter



Dafür benötigt man nun ein wenig Logik, um das Ganze anzusteuern. Die Idee ist hierbei, dass man aus Sicht eines Programmes zwei Bytes nacheinander schreibt, und diese beiden Bytes im Anschluss hieran durch die Hardware als ein 16 Bit-Wort an das IDE-Device übertragen werden.

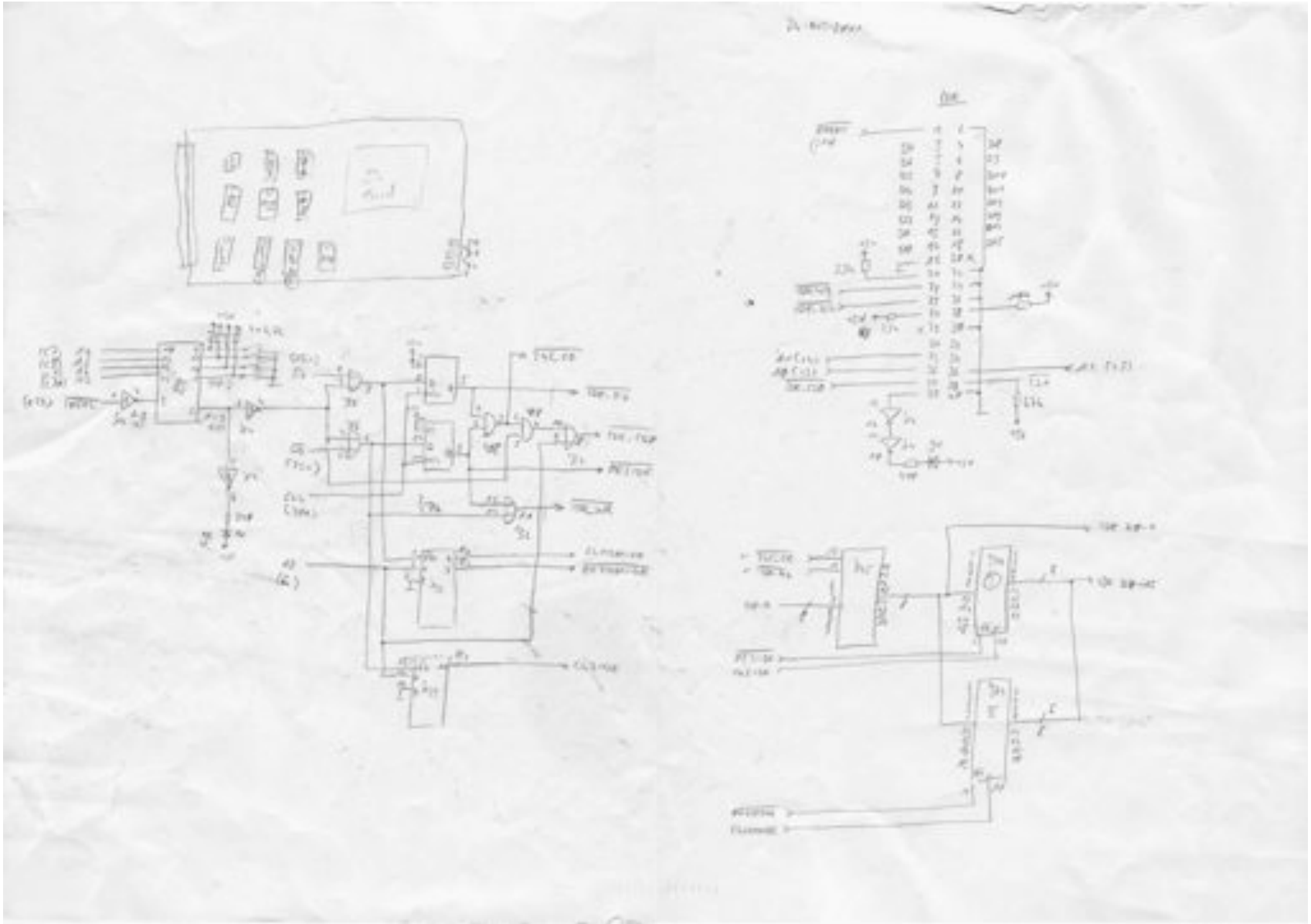
Dazu kommen noch ein paar Timingekligkeiten, so dass sich letztlich folgende Schaltung ergibt:


IDE_RD
IDE_CS#
'32
REZIDE
IDE_WR
CLKIDE

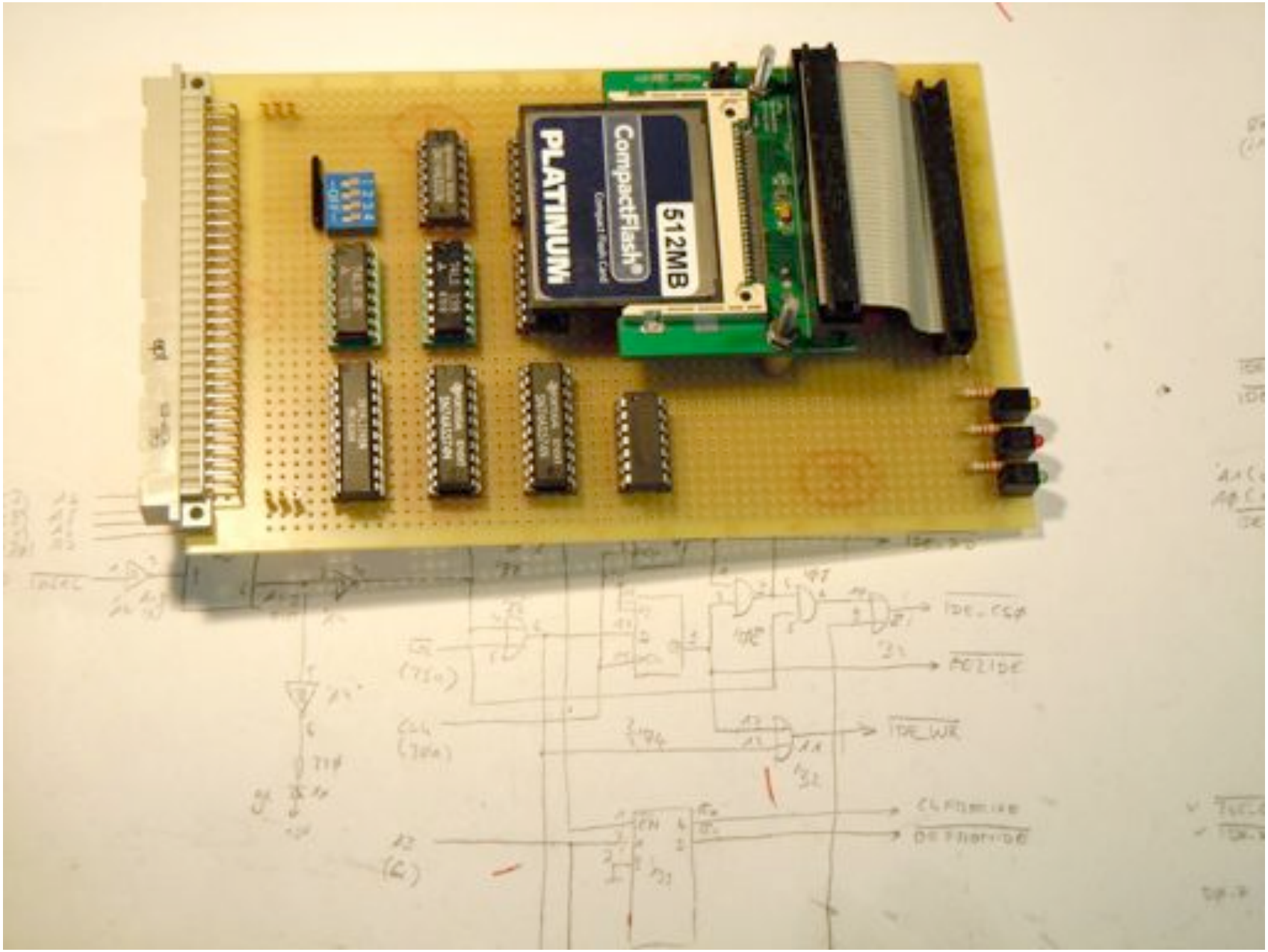
Wenn man nun alles zusammen nimmt, braucht man eine zweite Europakarte (160 mm $\times$ 100 mm).

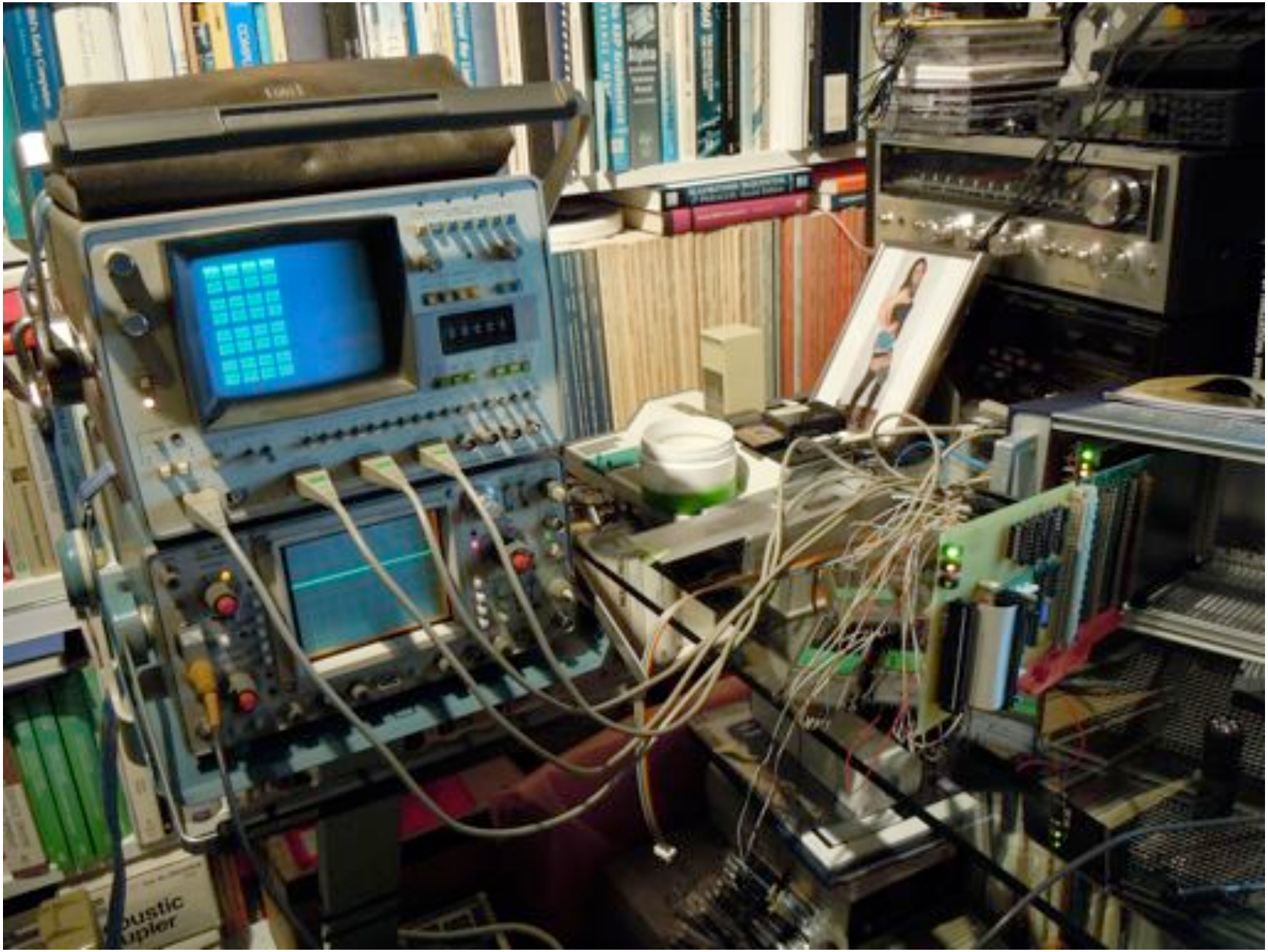
. . . und wenn man dann auch noch die IDE-Spezifikation missverstanden hat und die beiden /CS-Leitungen falsch beschaltet, muss man noch ein wenig Zeit mit Fehlersuche verbringen. :-)

Alles zusammen

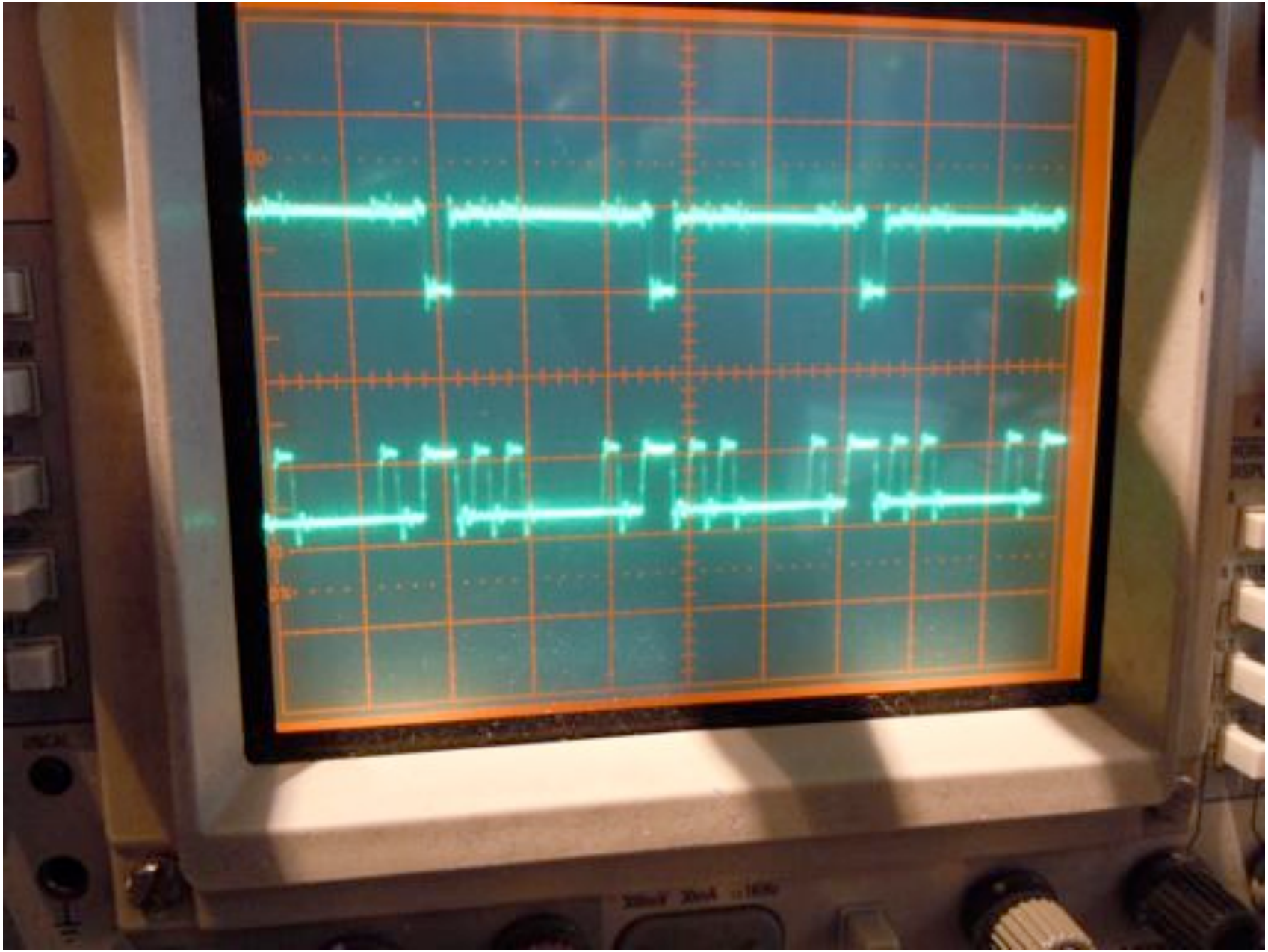


IDE-Controller



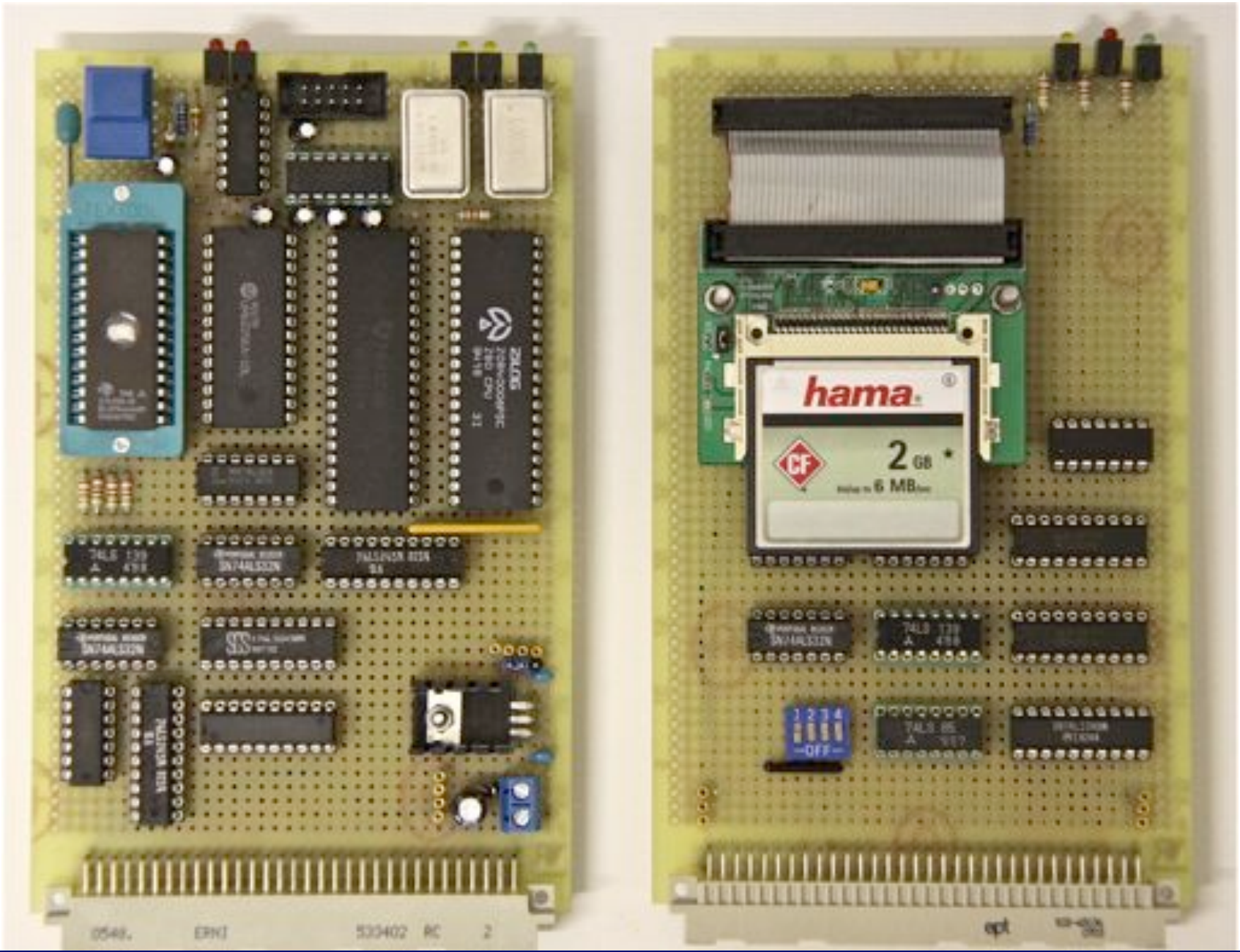


Zugriffe

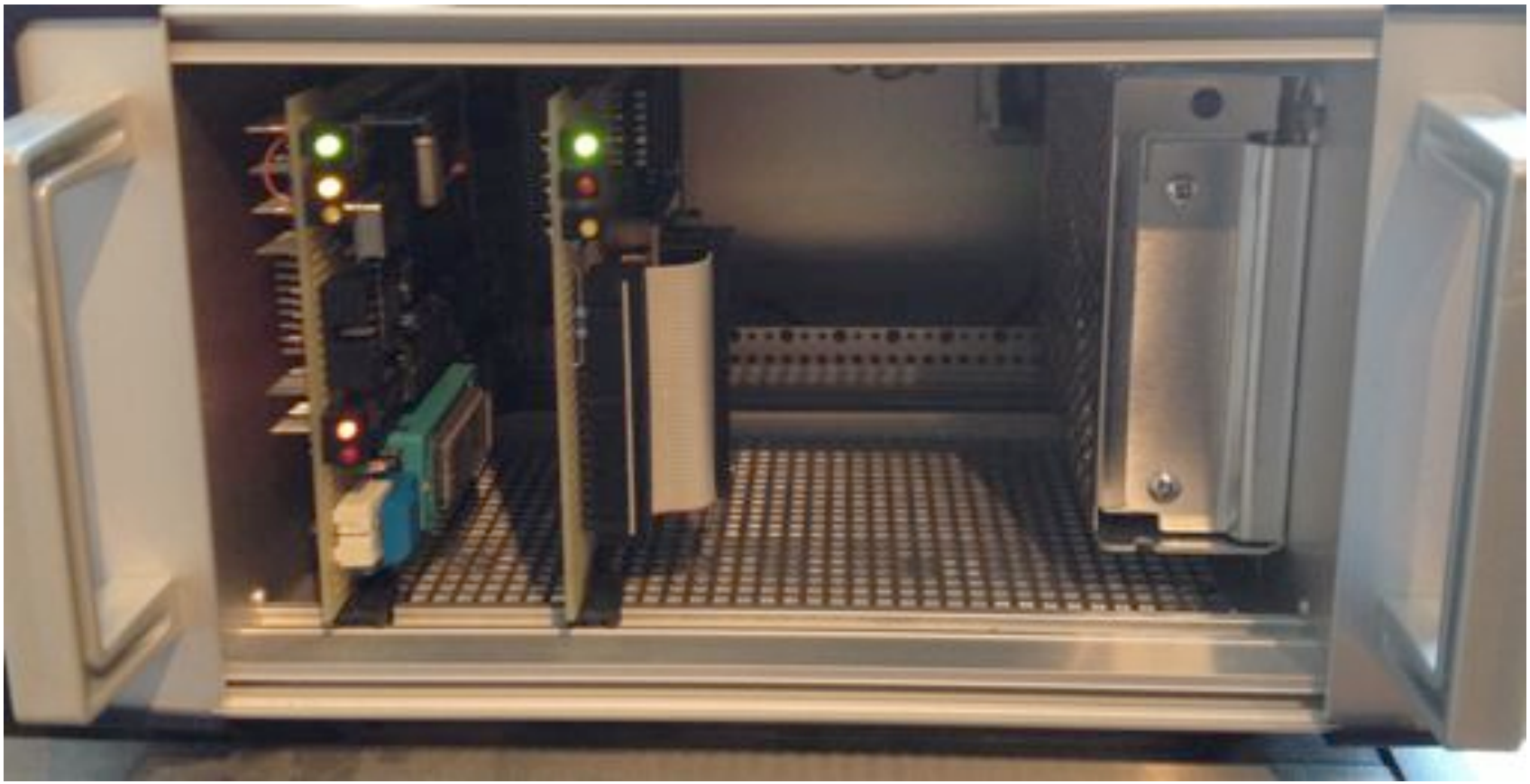


Gesamtsystem

Gesamtsystem



Die beiden Platinen werden nun noch in ein 10-Zoll-Gehäuse eingebaut und bekommen ein eigenes Netzteil. Links die CPU-Platine, rechts daneben der IDE-Controller mit CF-Adapter, ganz rechts das Netzteil:



Ein Minibetriebssystem

Nachdem nun Hardware vorhanden ist, braucht man ein Minibetriebssystem, mindestens also einen sogenannten *Monitor*. Welche Funktionalitäten muss dieser mindestens implementieren?

- „Devicetreiber“ (maßlos übertrieben)
- Initialisierungsroutinen für die Devices
- Einen einfachen Kommandointerpreter mit der Möglichkeit, Speicher zu laden, lesen, kopieren, dumpen, disassemblieren etc.
- Ein Filesystem, wenigstens ein lesendes, um Crossdevelopment machen zu können
- Idealerweise Interpreter für eine (zwei?) Programmiersprachen wie Forth und/oder BASIC

Ein Filesystem ist leichter gesagt als getan – welche Alternativen stehen zur Auswahl?

Eigenes Filesystem: Kann beliebig einfach gehalten werden, d.h. unter Umständen mit minimalem Entwicklungsaufwand umsetzbar. Nachteil: Für das Hostsystem, auf dem entwickelt wird, muss ein entsprechendes Filesystem implementiert werden, was wieder recht kompliziert ist.

CP/M-Filesystem: Bekannter Standard, allerdings nur sinnvoll, wenn man mit einem CP/M-System Daten austauschen möchte.

FAT16: Schon recht kompliziert, wird aber von allen gängigen Hostsystemen mehr oder weniger nativ unterstützt.

Wie sich schnell herausstellte, ist die Implementation eines FAT16-Filesystems in Assembler gar nicht trivial, stellenweise sogar recht herausfordernd.

Ohne die Hilfe meines Freundes, Herrn Dr. Ingo Klöckl, gäbe es vermutlich noch immer kein FAT16-Filesystem in Z80-Assembler. Das, was der Monitor verwendet, haben wir an einem langen Wochenende im Januar 2012 geschrieben.

Die gegenwärtige Implementation ist dennoch recht rudimentär:

- Unterverzeichnisse werden nicht unterstützt.
- Es sind nur Lesezugriffe möglich (was dennoch sehr hilfreich ist).

Mag jemand mitentwickeln? Mir fehlt irgendwie die Muße, daraus eine „richtige“ FAT16-Unterstützung zu machen. . . :-)

Welche Kommandos beherrscht das Minibetriebssystem mittlerweile?

```
Z> HELP: Known command groups and commands:

        C(ontrol group):
            C(old start), I(nfo), S(tart), W(arm start)
        D(isk group):
            I(nfo), M(ount), T(ransfer), U(nmount)
            R(ead), W(rite)
        F(ile group):
            C(at), D(irectory), L(oad), R(un)
        H(elp)
        M(emory group):
            (dis)A(ssemble), D(ump), E(xamine), F(ill), I(ntel Hex Load),
            L(oad), R(egister dump),
        S(ubsystem group):
            F(orth), B(ASIC)
```

Memorydump etc.

```
Z> MEMORY/DUMP: START=0000 END=0050
0000: F3 31 3B FB 18 17 00 00 C5 E5 DD E5 E1 29 01 A9   .1;..........)..
0010: 1D 09 4E 23 46 2B C5 DD E1 E1 C1 DD E9 3E 80 D3   ..N#F+.......>..
0020: 03 3E 0C D3 00 AF D3 01 3E 03 D3 03 3E 07 D3 02   .>......>...>...
0030: 21 5F 01 CD A8 12 3A 3C FB FE AA 28 18 21 11 03   !_....:<...(.!..
0040: CD A8 12 21 00 80 54 5D 13 01 FF 7F 36 00 ED B0   ...!..T]....6...
0050: 21 3C FB 36 AA 21 99 02 CD A8 12 CD 30 03 FE 43   !<.6.!......0..C

Z> MEMORY/DISASSEMBLE: START=0000 END=0010

0000 F3          DI
0001 31 3B FB    LD   SP,0FB3BH
0004 18 17       JR   1DH
0006 00          NOP
0007 00          NOP
0008 C5          PUSH BC
0009 E5          PUSH HL
000A DD E5       PUSH IX
000C E1          POP  HL
000D 29          ADD  HL,HL
000E 01 A9 1D    LD   BC,1DA9H

END OF DISASSEMBLER RUN.
```

Programm laden und ausführen

```
Z> DISK/MOUNT
        FATNAME:    MSDOS5.0
        CLUSIZ: 40
        RESSEC: 0008
        FATSEC: 00F0
        ROOTLEN:    0200
        PSIZ:       003C0030
        PSTART: 00001F80
        FAT1START:  00001F88
        ROOTSTART:  00002168
        DATASTART:  00002188
Z> FILE/DIRECTORY
Directory contents:
-----------------------------------------
FILENAME.EXT  DIR?   SIZE (BYTES)  1ST SECT
-----------------------------------------
ECHO    .EXE        00000032    00002188
S3      .4TH        000000B3    00002D08
SIN     .4TH        0000009F    00002D48
TEST    .4TH        0000001D    00002D88
TEST    .BAS        00000044    00003208
GUESS   .BAS        0000010A    000035C8
MANDEL  .ASM        00002812    00003C08
MANDEL  .EXE        0000019A    00003C48
```

Programm laden und ausführen

```
Z> FILE/RUN: FILENAME=mandel.exe
019A bytes loaded. Run...
```

```
Z> FILE/CAT: FILENAME=s3.4th
variable dt 1 dt !
: plot -1 do space loop 42 emit cr ;
: iterate tuck negate dt @ * 10 / + dup dt @ * 10 / rot + ;
: doit 1000 0 do iterate dup 4 / 40 + plot loop ;
100 0 doit

Z> SUBSYSTEM/FORTH

Z80 CamelForth v1.01  25 Jan 1995
  06-JUN-2012: Initial N8VEM port
  12-MAY-2013: Lower case conversion

load
Enter Filename: S3.4TH
                                                *
                                                  *
                                                    *
                                                      *
                                                        *
                                                          *
                                                            *
                                                              *
                                                                *
                                                                  *
```

```
10 X0=-2:X1=.5:Y0=-1:Y1=1:I1=20
20 X2=.04:Y2=.1:D$=" -+*=#&"
30 FOR Y=Y0 TO Y1 STEP Y2
40 FOR X=X0 TO X1 STEP X2
50 Z0=0:Z1=0
60 FOR I=1 TO I1
70 Z2=Z0*Z0-Z1*Z1:Z3=2*Z0*Z1
80 Z0=Z2+X:Z1=Z3+Y:IF Z0*Z0+Z1*Z1>4 THEN GOTO 100
90 NEXT I
100 A=SQR(Z0*Z0+Z1*Z1):I=INT(A)-7*INT(A/7)+1:PRINT MID$(D$,I,1);
110 NEXT X:PRINT
120 NEXT Y
run
++++++++++++###=====++++++++++++++#==++++++=++++++++++++++=++++++
++++++++++##=====++++++++++++++++==++++++++++++++ ++=+++=+++++=+++
++++++++#====++++++++++++++++++++++++++=++=++    -++++++++++++++
++++++====++++++++++++++++++++++++++++++++++     ++++++++++++++
++++==++++++++++++++++++++=+++=++++   +              +=+ +-+++
++++++++++++++++++++++++++++++++++++                    ++++
++++++++++++++++++++=++++++++++=+-                       ++++
+++++#=+++++#++++++++  ++ +=++++                         +++
++#=++++++++++++++++         ++                           ++
++++++++=+++=+++=+=                                      +++
+ -- -- - -                                            +++=++
++++++++=+++=+++=+=                                      +++
++#=++++++++++++++++         ++                           ++
+++++#=+++++#++++++++  ++ +=++++                         +++
++++++++++++++++++++=++++++++++=+-                       ++++
++++++++++++++++++++++++++++++++++++                     ++++
++++==++++++++++++++++++++=+++=++++   +              +=+ +-+++
++++++====++++++++++++++++++++++++++++++++++     ++++++++++++++
++++++++#====++++++++++++++++++++++++++=++=++    -++++++++++++++
++++++++++##=====++++++++++++++++==++++++++++++++ ++=+++=+++++=+++
Ok
```

Um einen kleinen Eindruck von der Komplexität selbst eines so einfachen Betriebssystemes zu geben, sind im folgenden die Anzahl von Codezeilen (LOC) für verschiedene Monitorversionen aufgelistet:

Version	Datum	LOC
1	28.09.2011	282
2	29.09.2011	418
3	02.10.2011	726
6	01.11.2011	1034
7	03.11.2011	1320
8	14.01.2012	2456
14	05.06.2013	3118

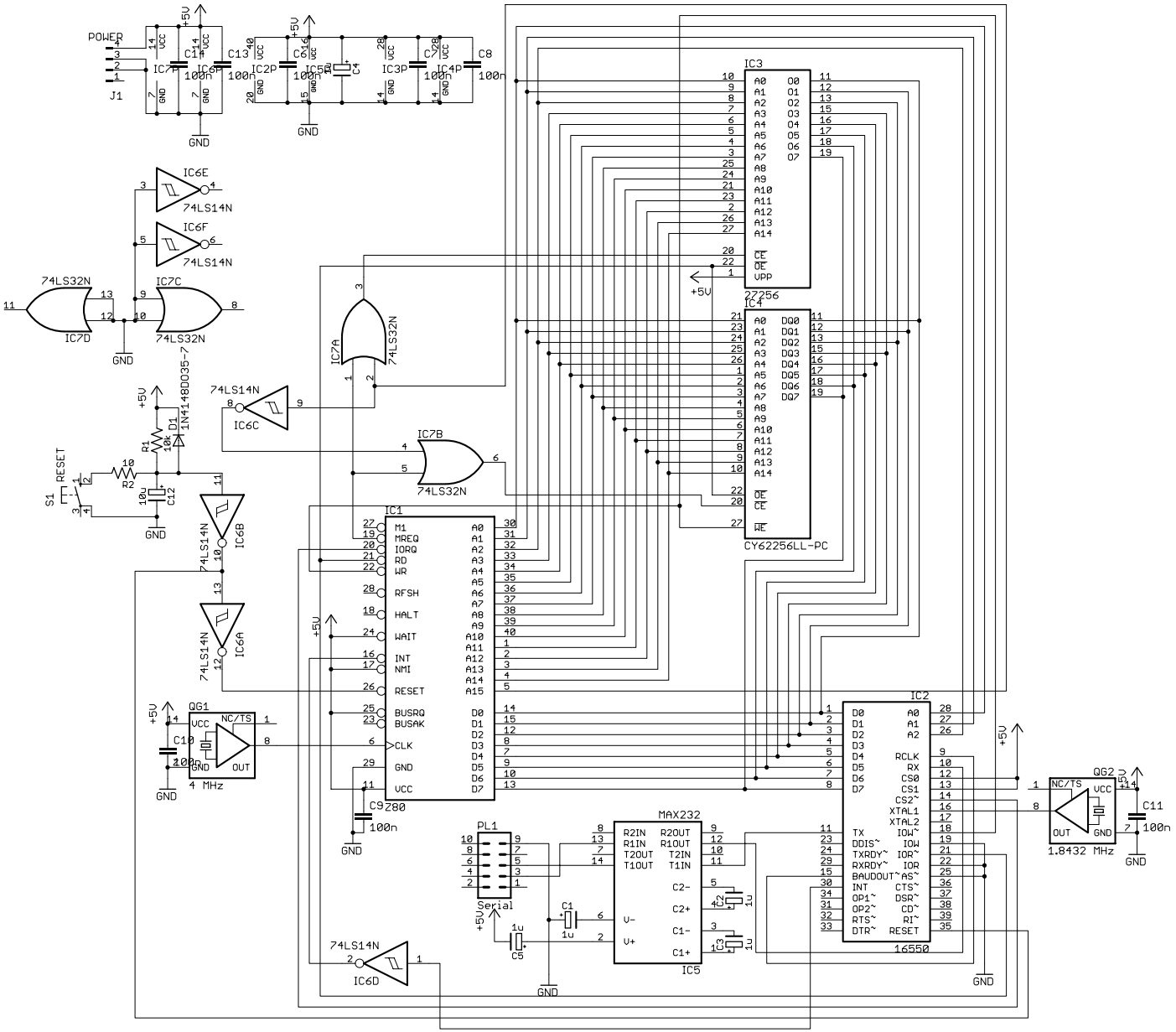
Die letzte Variante des Monitors beinhaltet noch einiges an Fremdsoftware: 4037 Zeilen CAMEL-Forth, 690 Zeilen Z80-Disassembler sowie 4508 Zeilen BASIC-Interpreter; in Summe also 12353 Zeilen Assemblercode.

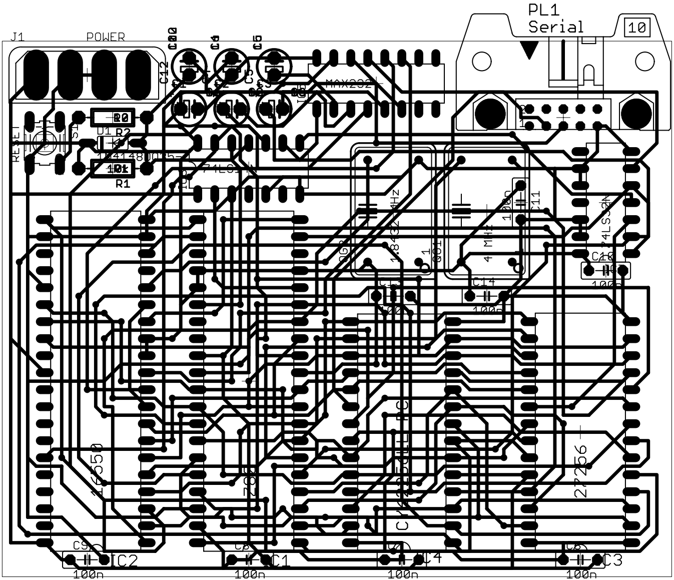
Minirechner

Das Ganze hat jedoch einen Nachteil: Der Nachbau erfordert einiges an Erfahrung und Geduld und Muße. Das muss (ohne IDE-Interface) auch einfacher (und preiswerter) zu machen sein...

Grundlage ist der zu Beginn beschriebene Wire-Wrap-Prototyp, der seiner Busschnittstelle beraubt wird, um mit nur einer halben Europakarte an Fläche auszukommen.

- Damit bleibt zwar nur die serielle Schnittstelle als Interface, was aber für Lehr- und Ausbildungszwecke ausreichend ist, solange der Monitor entsprechende Uploadmöglichkeiten bietet.
- Der Rechner kann problemlos über Stunden hinweg aus einem USB-Akku betrieben werden, was für Vorführungen ausgesprochen praktisch ist.
- Die Gesamtkosten liegen bei unter 50 EUR.


POWER
J1
+5V
GND
C14
C13
100n
C16
C15
10u
C8
IC7P
IC6P
IC2P
IC5P
IC3P
IC4P
IC6E
IC6F
74LS14N
74LS32N
IC7C
IC7D
IC7A
IC7B
IC6C
IC6B
IC6A
IC6D
1N4148D035-7
D1
R1
10k
R2
10
S1
RESET
C12
10u
IC1
Z80
M1
MREQ
IORQ
RD
WR
RFSH
HALT
WAIT
INT
NMI
RESET
BUSRQ
BUSAK
CLK
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
D0
D1
D2
D3
D4
D5
D6
D7
C9
100n
IC3
27256
IC4
CY62256LL-PC
QG1
4 MHz
C10
100n
IC2
16550
D0
D1
D2
D3
D4
D5
D6
D7
RCLK
RX
CS0
CS1
CS2~
XTAL1
XTAL2
TX
DDIS~
TXRDY~
RXRDY~
BAUDOUT~
INT
OP1~
OP2~
RTS~
DTR~
IOW~
IOW
IOR~
IOR
AS~
CTS~
DSR~
CD~
RI~
RESET
QG2
1.8432 MHz
C11
100n
PL1
Serial
MAX232
IC5
R2IN
R1IN
T2OUT
T1OUT
R2OUT
R1OUT
T2IN
T1IN
C2-
C2+
V-
V+
C1-
C1+
C1
C2
C3
C4
1u



PL1
Serial
10
J1
POWER
MAX232
74LS14N
R1
R2
D1
16550
Z80
27256
74LS32N
4 MHz
C14
100n
C9
IC2
IC1
IC4
IC3


ICL232CPE
H9501
Z8400BB1
Z80BCPU
88346 -ITALY
4.000MHz

Was gibt es noch?

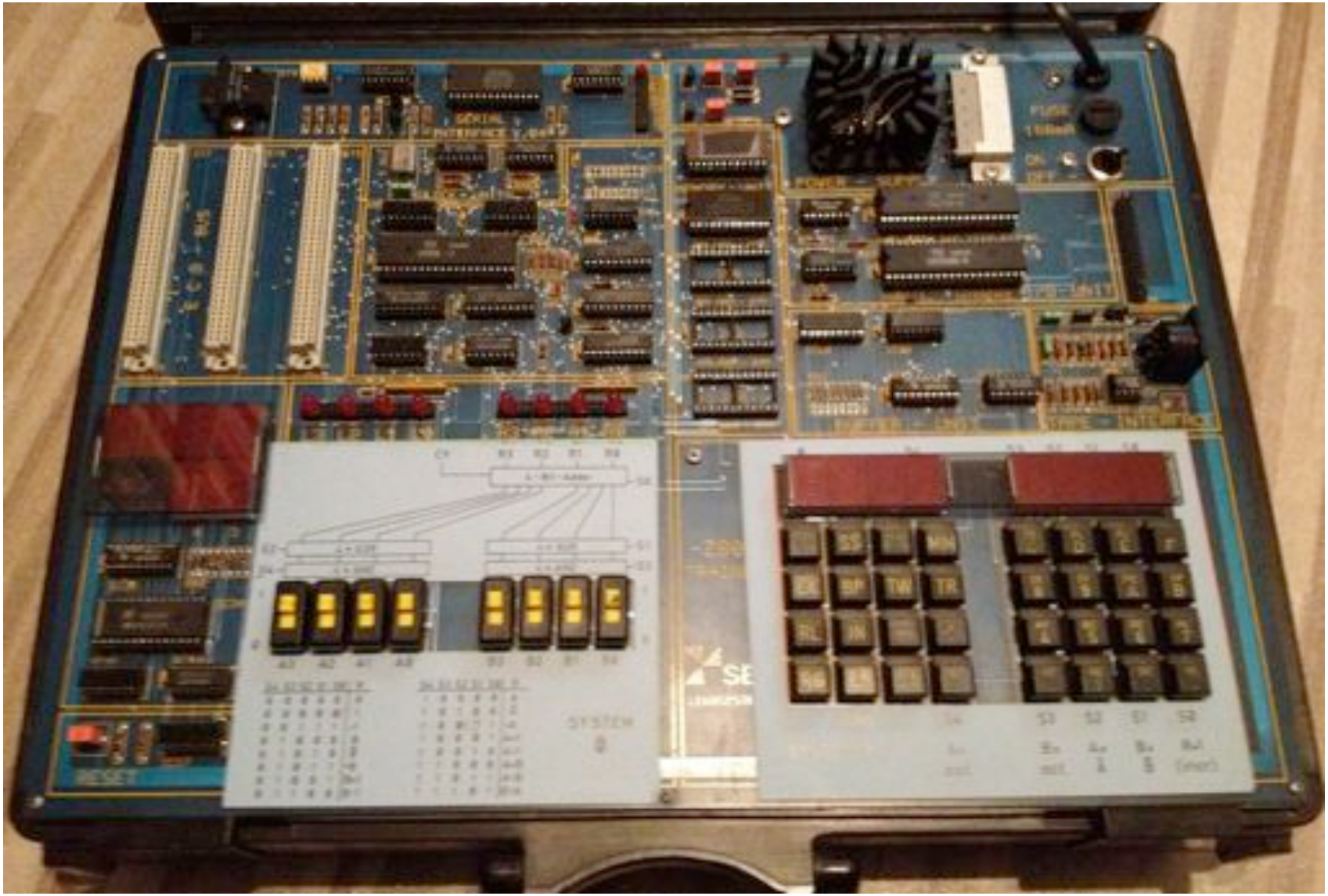
Natürlich kann man auch historische Lehr- und Lerncomputer verwenden, wobei hier jedoch der Spaß an der Entwicklung und am Aufbau entfällt.

Schöne Systeme gibt es auch als Replicas von `http://www.brielcomputers.com/`, wenn man nichts Historisches erwerben mag oder kann...

Im Folgenden ein paar Beispiele:

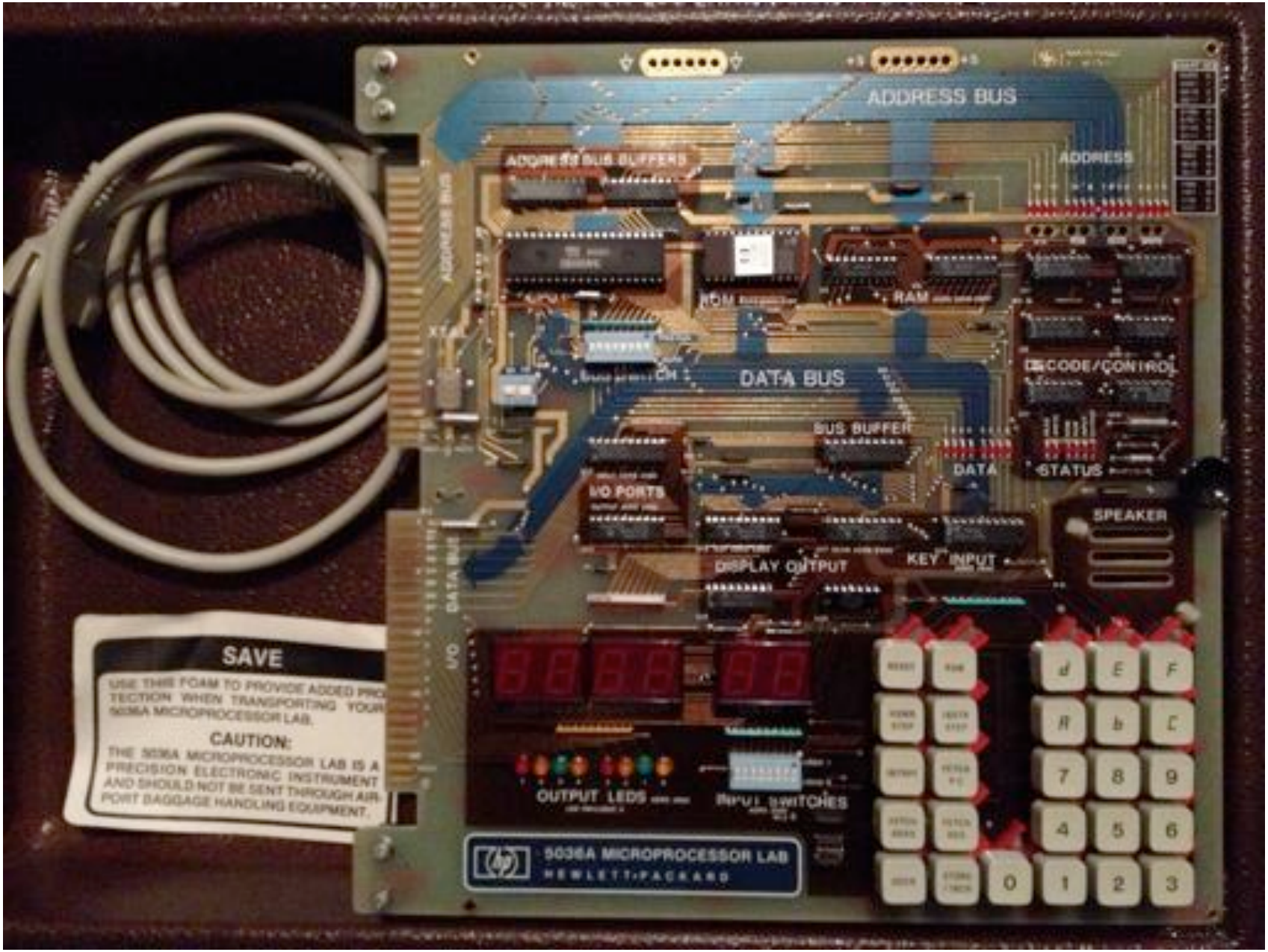

MICRO-PROFESSOR
MPF-I Handbuch
MPF--1

SEL Z80-Trainer




PROFI-5/50 E

HP 5036A


ADDRESS BUS
ADDRESS
DATA BUS
RAM
BUS BUFFER
DATA
STATUS
I/O PORTS
SPEAKER
DISPLAY OUTPUT
KEY INPUT
OUTPUT LEDS
INPUT SWITCHES
5036A MICROPROCESSOR LAB
HEWLETT-PACKARD
SAVE
CAUTION:


Micro-KIM
SST
GO
ST
RS
AD
DA
PC
6502 CPU
www.brielcomputers.com

Anfang 2012 fand ich das N8VEM-Projekt von Andrew Lynch,[2] in dessen Zentrum ebenfalls ein Z80-Einplatinenrechner (mit ECB-Bus-Interface) steht.

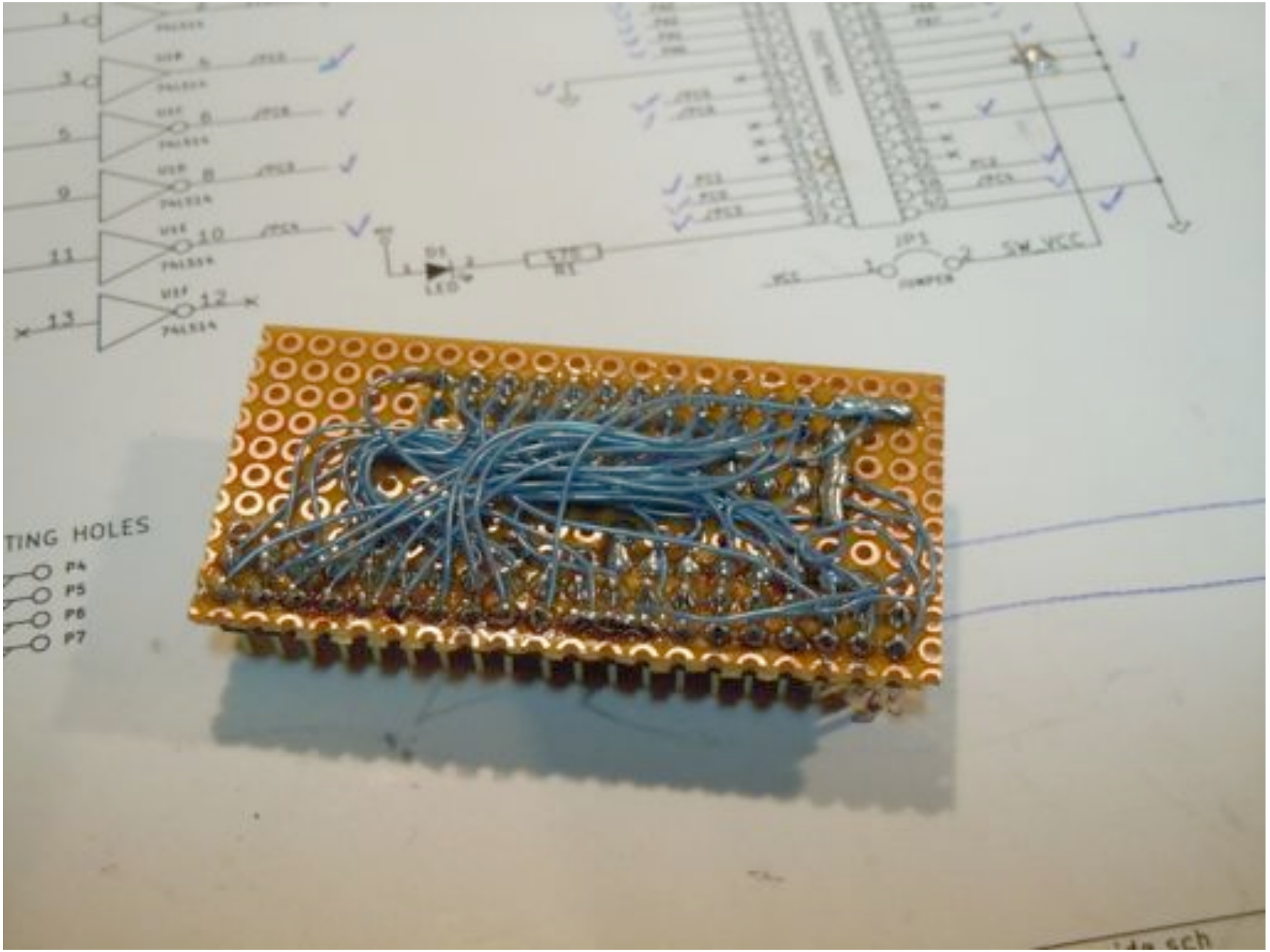
Die sehr professionell umgesetzte Leerplatine ist für ca. 20 EUR bei Andrew Lynch erhältlich – die Bestückung bereitet keine Schwierigkeiten, so dass man als geübter Löter nach ca. 2 Stunden einen funktionsfähigen Z80-Rechner sein eigen nennen darf:

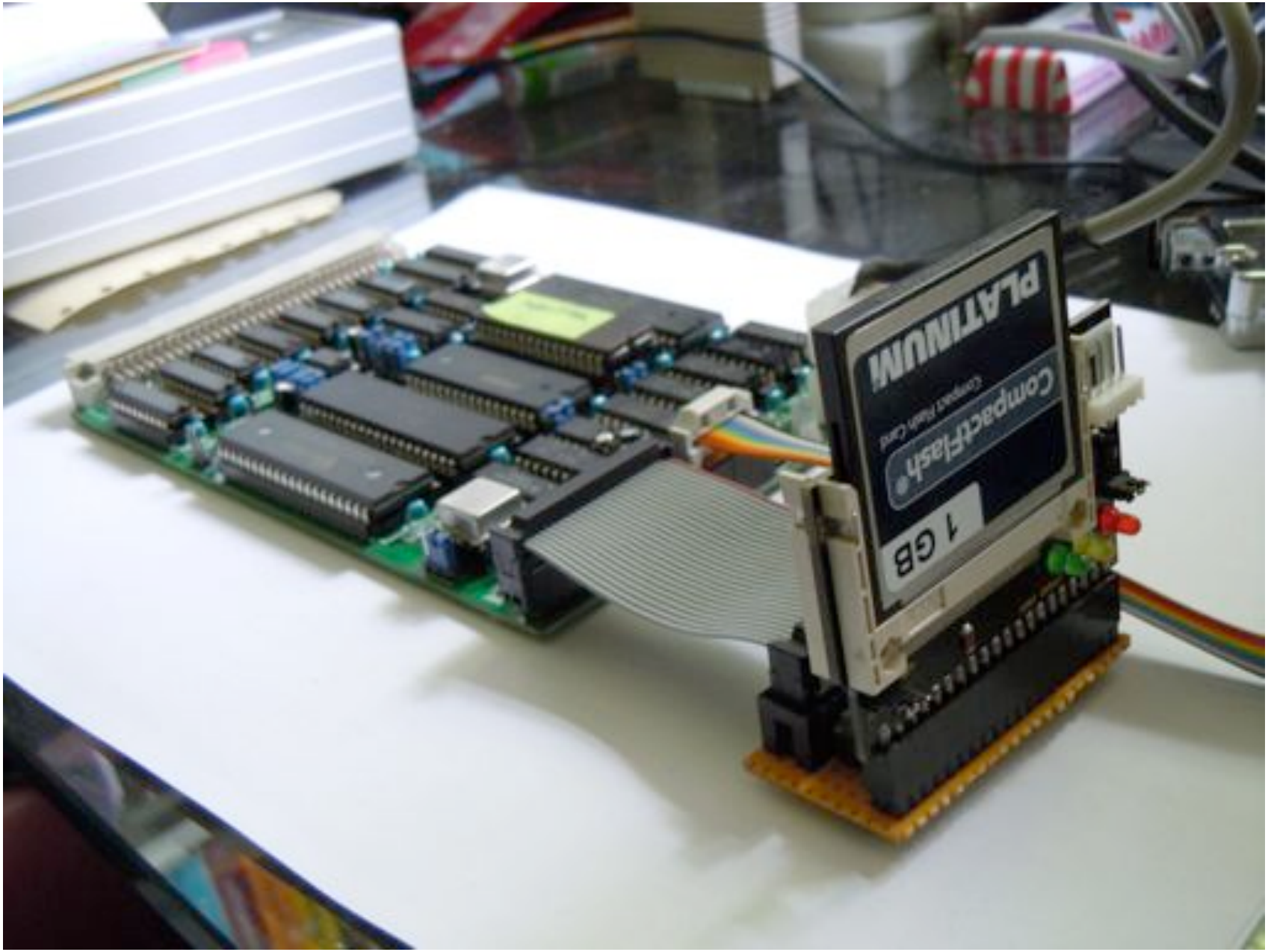
[2] Siehe `http://n8vem-sbc.pbworks.com`, Stand 08.06.2013.


74LS 174
74LS 174
74LS 174
74LS 139
N8VEM

Die N8VEM-Karte beinhaltet einen 8255-Interfacebaustein, der drei jeweils 8 Bit breite Ports zur Verfügung stellt. Diesen Baustein nutzt die N8VEM-Community (unter anderem) als IDE-Schnittstelle, indem das, was der zuvor beschriebene IDE-Controller in Hardware macht, in Software erledigt wird (was natürlich deutlich langsamer als die zuvor beschriebene Hardwarelösung ist).

Als einzige Erweiterung wird ein einfacher Adapter mit einem sechsfach Inverter benötigt, für den es eine vorgefertigte Platine gibt, der aber in Ermangelung einer solchen oder nagender Ungeduld auch schnell selbst implementiert werden kann:

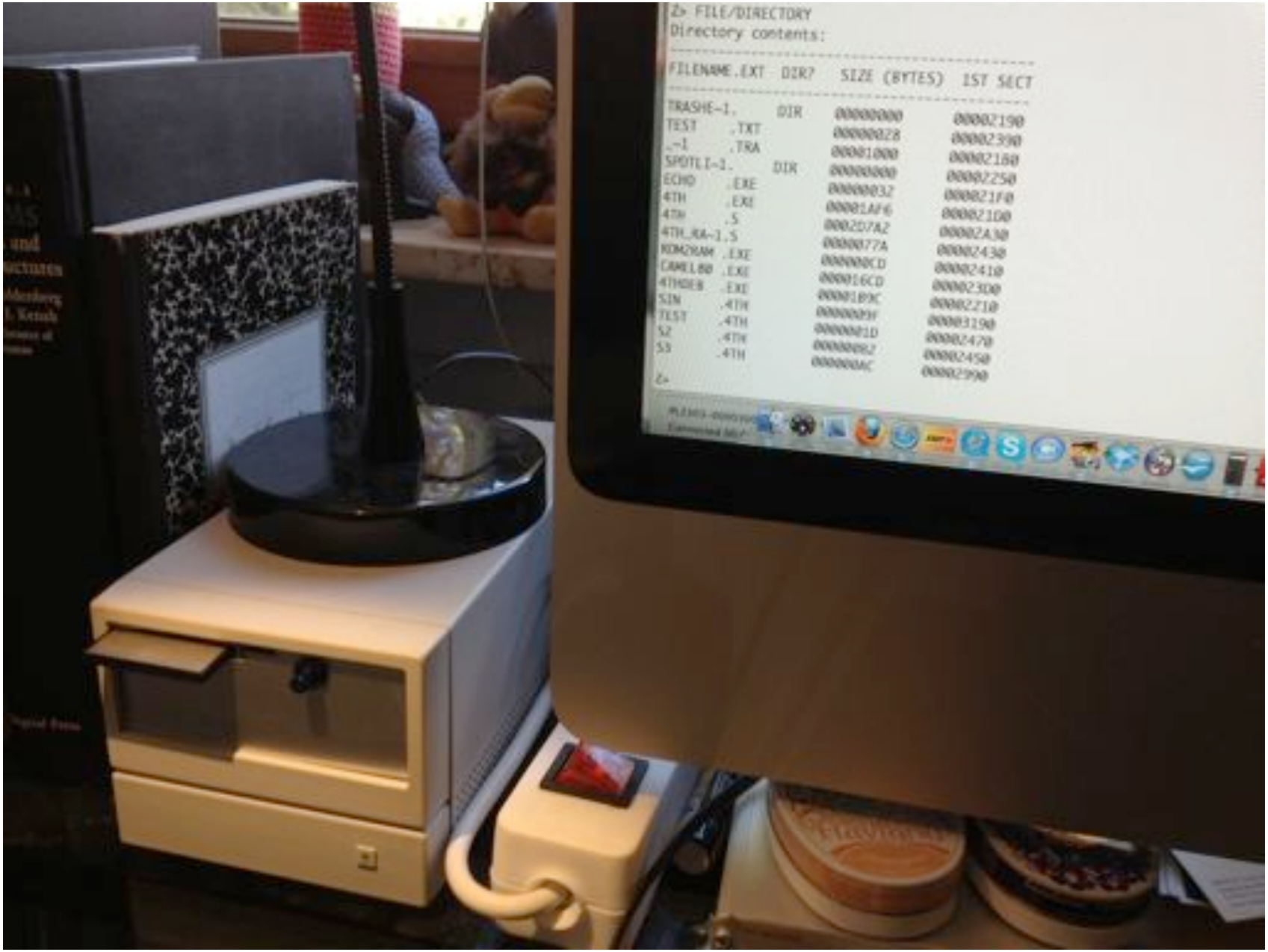

TING HOLES
P4
P5
P6
P7


PLATINUM
CompactFlash
1 GB

Das Portieren des Monitors auf die N8VEM stellte sich als recht einfach heraus – zu beachten waren im Wesentlichen die folgenden Punkte, was etwa einen Tag in Anspruch nahm:

- Unterstützung einer einfachen Bankswitching-Logik (1 MB EPROM, 512 kB RAM sind Standard).
- Andere Adresslage der Controllerbausteine.
- Völlig andere IDE-Implementation.

Nachdem alles funktionierte, wurde natürlich auch ein Gehäuse mit Netzteil benötigt, wobei zufällig ein altes DAT-Laufwerksgehäuse zur Verfügung stand:


Z> FILE/DIRECTORY
Directory contents:
FILENAME.EXT DIR? SIZE (BYTES) 1ST SECT
TRASHE~1. DIR 00000000 00002190
TEST .TXT 00000028 00002390
~1 .TRA 00001000 00002180
SPOTLI~1. DIR 00000000 00002250
ECHO .EXE 00000032 000021F0
4TH .EXE 0001AF6 00002100
4TH .S 0002D7A2 00002A30
4TH_RA~1.S 0000077A 00002430
KOMZRAM .EXE 000000CD 00002410
CAMEL80 .EXE 000016CD 00002300
4THDEB .EXE 0001B9C 00002210
SIN .4TH 0000009F 00003190
TEST .4TH 0000001D 00002470
S2 .4TH 00000002 00002450
S3 .4TH 000000AC 00002990
Z>

Ausblick

Ausblick

. . . oder auch „lessons learned“:

- Einen kleinen 8 oder 16 Bit Rechner wirklich selbst zu bauen, macht Spass und ist keine unüberwindbare Herausforderung.
- Einen Monitor zu schreiben ist schon komplexer, ein Filesystem, und sei es nur FAT16, zu implementieren, ist bereits wirklich schwierig.
- Vielleicht findet der Minirechner etwas weitere Verbreitung – Leerplatinen können bei üblichen Leiterplattenherstellern für ca. 15 EUR/Stück gefertigt werden, die Software ist vom Autor erhältlich (selbstverständlich kostenfrei).

Gesucht...

Gesucht werden Mitstreiter! :-) Vor allem auf der Softwareseite:

- Erweiterung des Filesystems, um auch Unterverzeichnisse und schreibenden Zugriff zu unterstützen.
- Implementation eines eigenen BASIC-Interpreters.
- Implementation oder Portierung (SymbOS?) eines multitaskingfähigen Betriebssystems für den Z80.
- Etc...

Der Autor kann unter folgender Mailadresse erreicht werden:

`ulmann@vaxman.de`

Vielen Dank für Ihre Aufmerksamkeit!