Creating my personal "superspec".

A project for the purpose of study. RoKo2012

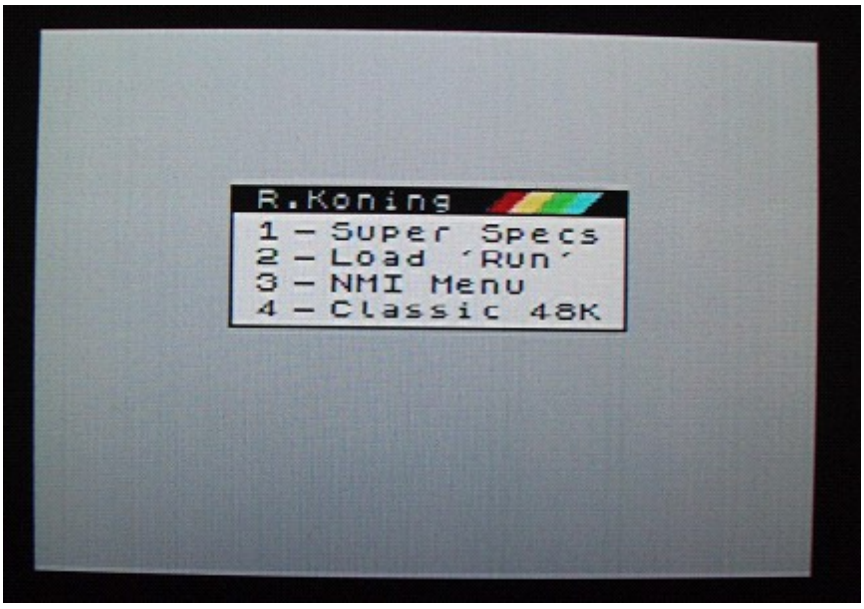
==========

Introduction.

==========

This is a project which I started as early as 1987. In those days the so called "dot commands" were a hot item on the Sinclair Spectrum. These were new commands that could be added to the BASIC. The dot was added to the name of the command to provoke a call to the error routine, from where special programming took care of the new commands. All sorts of toolkits were around, with Andrew Wrights BetaBasic and Mike Leamans MegaBasic up front.
Being a hardware man myself, I at that moment already had extended my hardware with extra 'ROM' banks, and also had invented my own method to add new keyword modes (modi for some), and new commands and functions to the Spectrums system. The commands BRIGHT, OVER and INVERSE only allow for the parameters 0 or 1, so I decided to use CHR$ 21 (BRIGHT) as a kind of 'Escape' and create my extensions using the parameters 128 and up. It appeared that the system accepted my new mode's and keywords in a rather natural way. From that moment on there were hundreds of examples around to learn (and borrow) from.
As a result of all the time that I spent playing with the matter, I never came to using the Basic extensions but got addicted to programming in Machine Code. Through the years a number of 'spin-off' projects were published in the (dutch) SGG Bulletin and the (german) SPC magazine.
My original goal was to take part in creating a 'better' Spectrum, but in a rather early stage it became clear to me that the PC had won the race and Windows along with it, and that my project never would go beyond the status of a silly dream of some stubborn person. This made no real diference for me as I had always considered the status of the Spectrum as a challenging gadget for studying and practicing programming skills.

Over the years my project has gone through many stages in hard and software and it probably is not finished yet. Anyway, today (2012, Speccy's 30th anniversary) is the time that I put bits and pieces from all kinds of experiments together in order to consolidate the results sofar. I think that the result is ready for service, or at least useful.
As far as the name 'superspec' (mentioned above) brings associations with a "Super Spectrum" for a "Superman", this is only ment as pun. In fact the name means 'super specifications'. Which just means that the specifications of this machine go beyond those of the standard Spectrum.



The hardware contours.

1. Video output combined with TV signal, over the existing VHF connector. Increased decoupling of +5V and + 12V for a better, cleaner picture.
2. A switching regulator is replacing the standard 7805 voltage regulator, in order to get rid of the enormous heatsource. Also for gaining space inside a rubber Spectrum.
3. Added polarity protection from the power supply by means of a diode.
4. Mounting the 48K RAM chips on sockets and replace these by type 4164. Thus creating 2 banks of 32K RAM. Some 16K of the extra RAM is needed for all kind of buffers and extra system variables.
5. Mounting the standard 16K ROM on a socket and replace it by a 64K EPROM (27C512), or by a 128K FlashRAM. (29EE010)
6. Wiring a programmed GAL22V10 inside the Spectrum, for providing the necessary logic for the added hardware.
7. Wiring a CompactFlash-card inside the Spectrum. (4GB expected)

The software contours.

The software resides in a ROM that holds 4 banks of 16K:

1. A modified Spectrum ROM that communicates with banks 2 & 3.
2. A ROM holding some 40 extra commands and 30 extra functions.
3. A ROM holding a complete Disk Operating System.
4. The original Spectrum ROM.

The easiest way for giving a general idea of what can be expected, is saying that several commands and functions known from Beta Basic can be found, and most of the disk commands as known from Opus Discovery. However, don't expect existing programs for these two systems to work, as no effort is made to keep or to create compatibility. This system is just ment to be close to how I feel that a Spectrum should be.

Missing at the moment (2012):

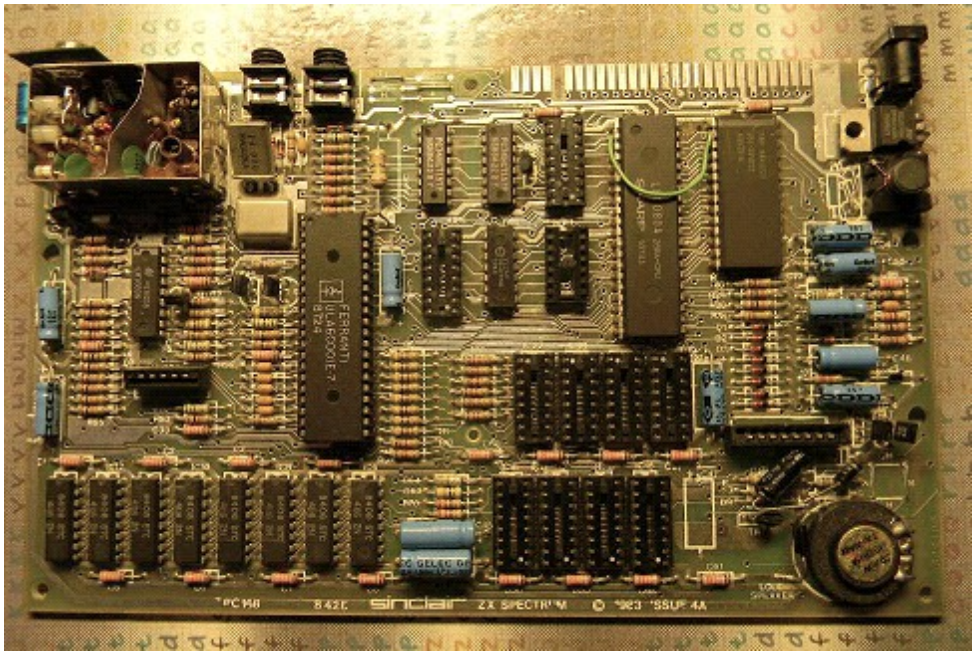
a. a hardware reset button.
b. NMI menu, RS232: hard + soft
c. Programming Manual

Credits.

A few names must be mentioned: Toni Baker for writing Mastering Machine Code, being the only book I found usefull. Steven Vickers for writing a genius BASIC dialect. Ian Logan & Geoff Wearmouth for their commented ROM disassemblies. Dave Corney for writing the Opus Discovery DOS. Andrew Wright for the glorious BetaBasic. A.Collier for repairing a calculation problem in ROM. Alvin Albrecht for his 'flood fill'. Rudy Biesma & Edwin Blink for the many enlightening brainstorming sessions that I had with them. I have to leave hundreds (if not thousands) of names out simply because I learned from almost everyone who ever wrote a contribution in one of the many computer magazines of the eighties...

The building of my second prototype. (nr.2)
==========

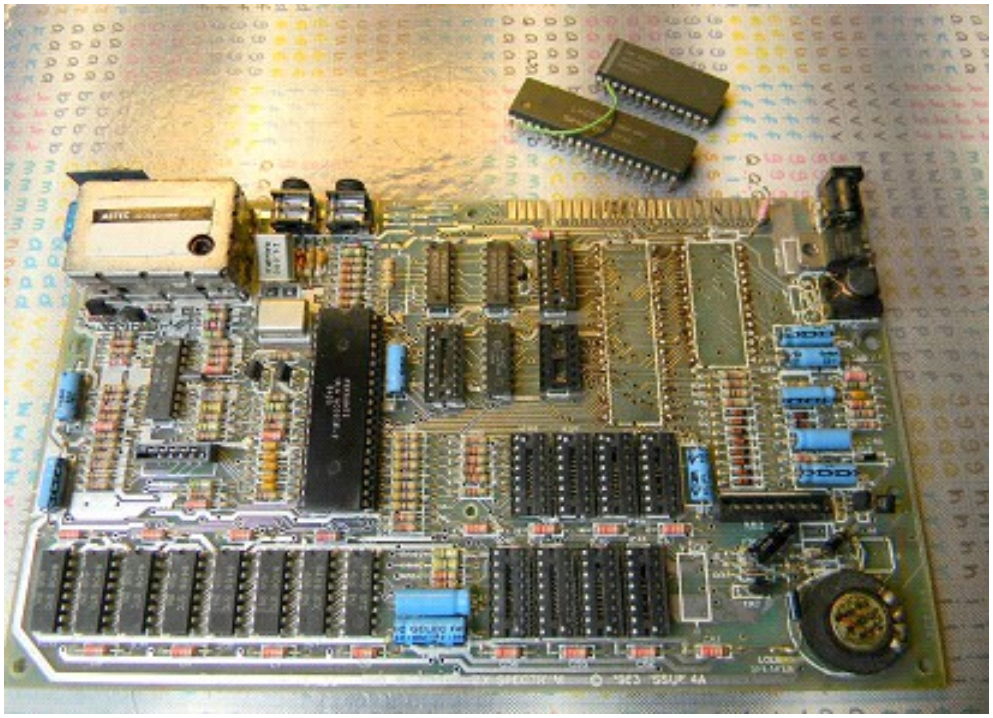
I start off with a circuit board of a 48K Spectrum issue 4A. As it happens the Z80 on this board is defect, and the complete 48K RAM extension is in sockets, except the 74LS00.



unmodded

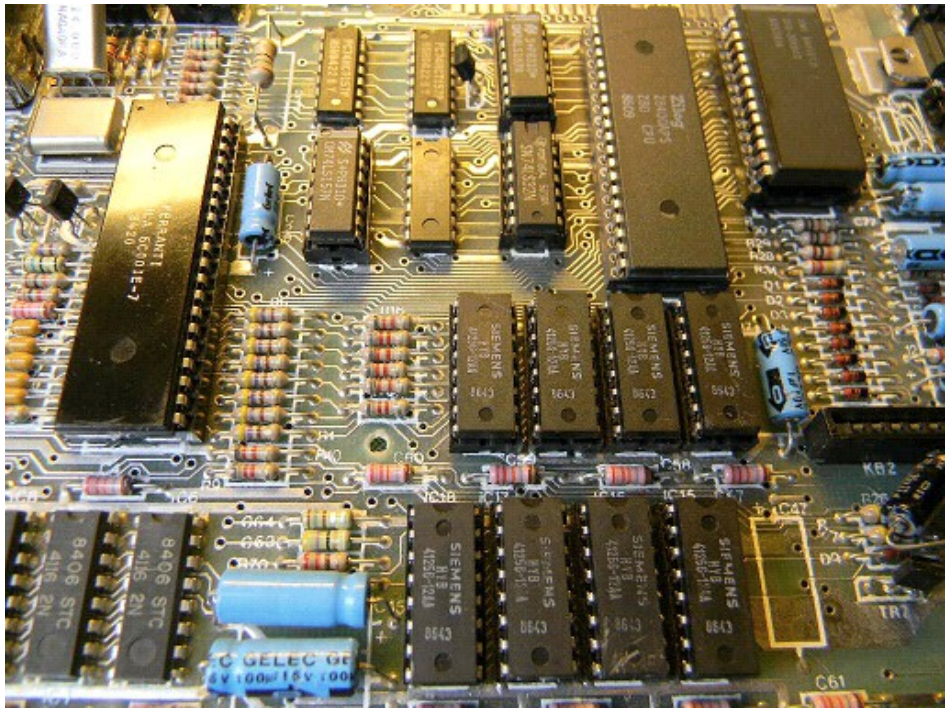
ADDING SOCKETS

Should the upper-RAM chips (3732) not have been in sockets, it would have been necessary to remove these as larger chips (4164) are needed. Now I only remove the Z80 chip for repairing, and also the ROM chip as it has to be socketed and replaced... The Z80 and ROM had been soldered before, and the resin-flux that was left has now turned brown in the removing process. Alcohol was used to clean up.



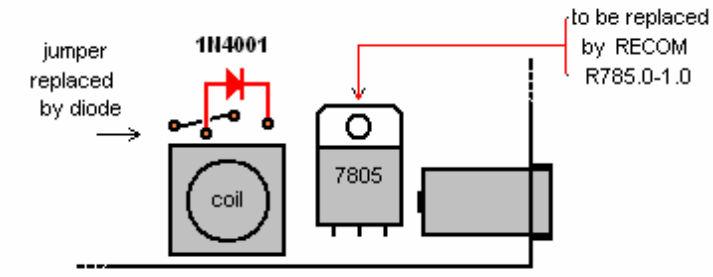
empty board

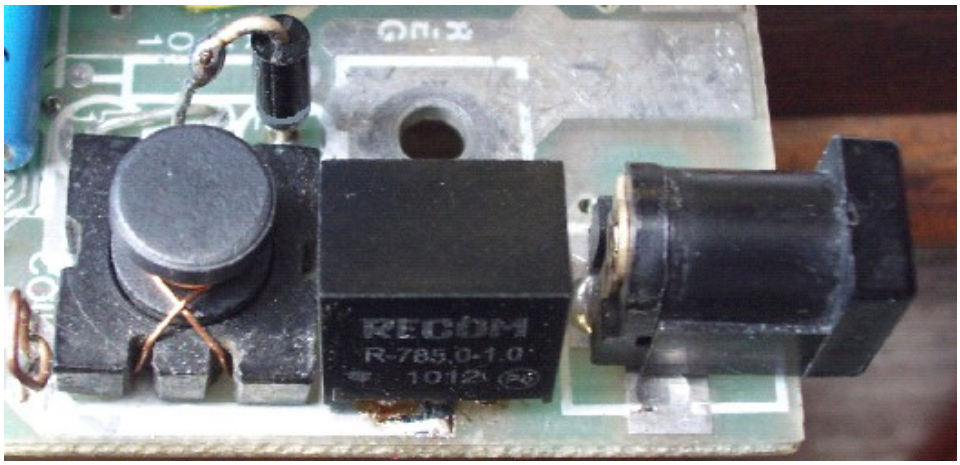
After soldering in the sockets, and inserting Z80 and ROM, the board is tested for working as 16K. The memory sockets are then filled with 4164s, and another test is done. I make a test after each step taken, so that in case of problems there only is one suspected issue. When everything is allright the most difficult part is done now. BTW. the 256K Siemens chips shown here did not refresh the 'alternative' bank.



allsockets

EXCHANGING 7805



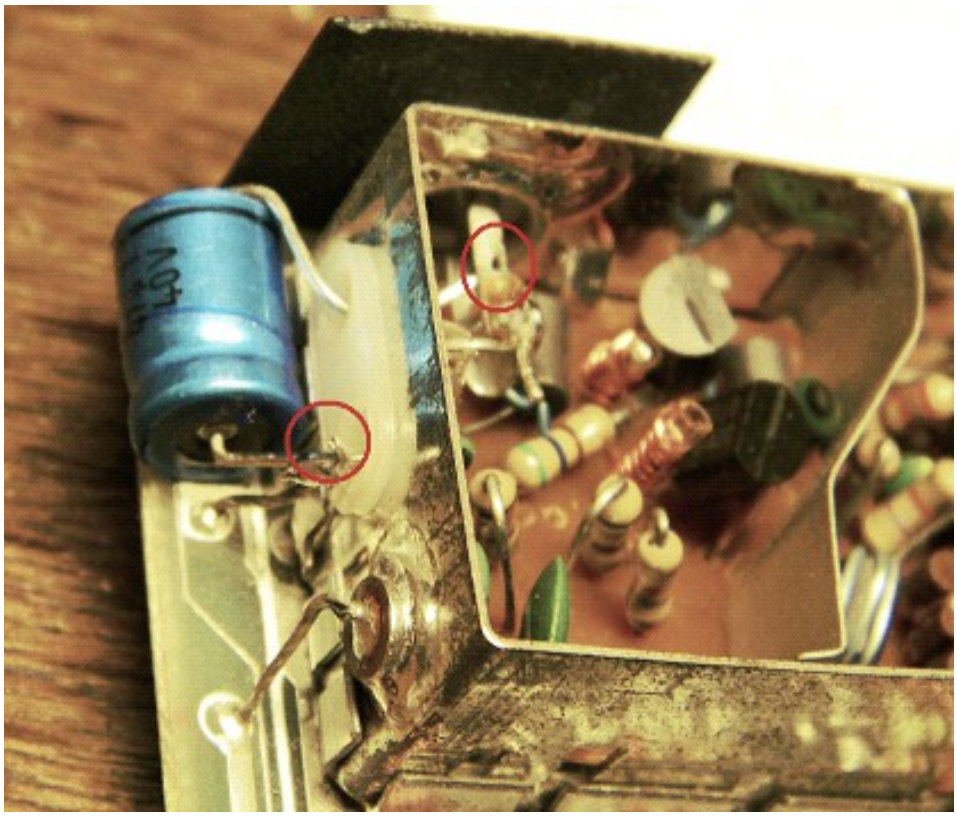


regulator+diode Note the direction of the diode.

As is illustrated on the previous page, I replace the 7805 by a R785, (switching regulator) and add a safety diode in the +9V line that comes from the connector. In the issue4 this is easily done. The chance to accidently connect an alien power brick exists, and a different polarity might kill the Spectrum. The diode is there to prevent such.
Using this Recom R785 makes the heatsink redundant, and the removal frees some valuable space inside rubber Speccies.

VIDEO OVER VHF

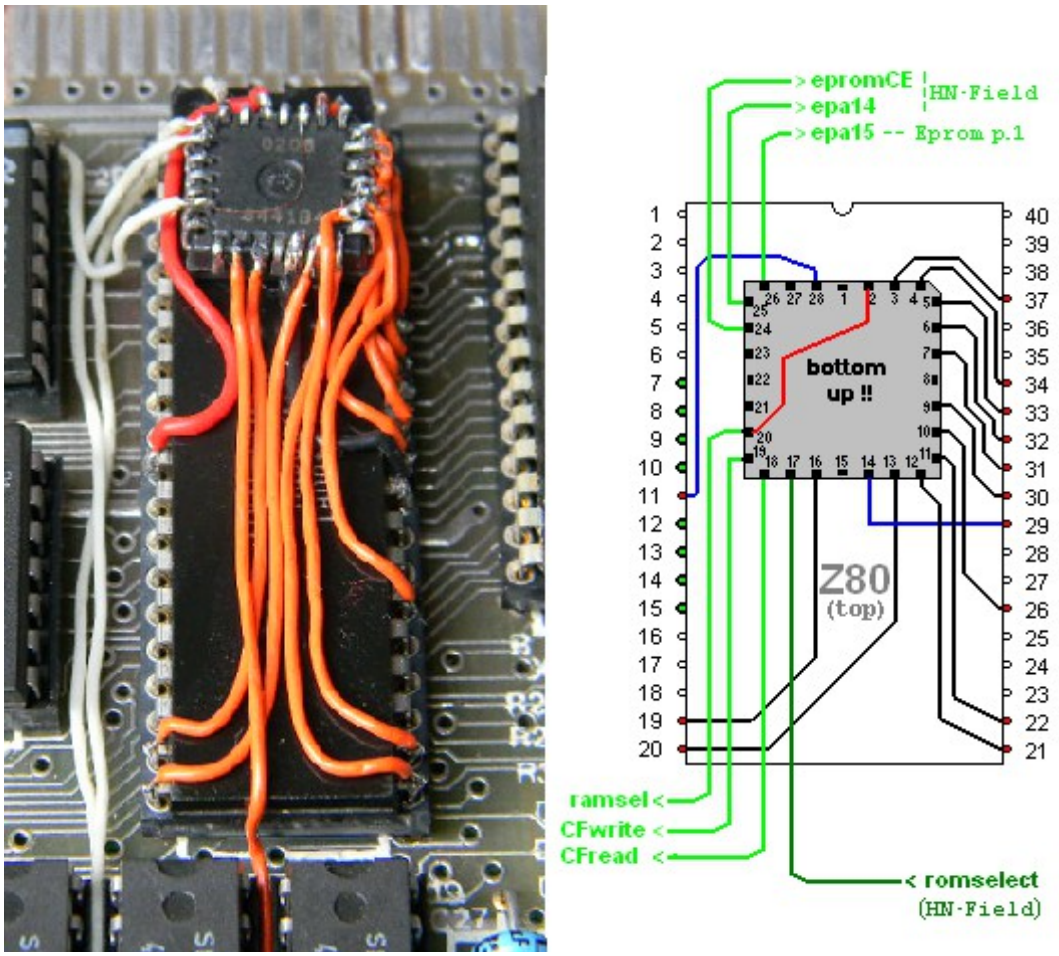
Next step is adding the video signal to the VHF output. This is done by connecting 'video' to the Cinch output connector on the modulator by means of a 47 uF electrolytic capacitor, minus at the output. This combined use of one connector is a modification Sir Clive himself would like. If he only knew about it.
Testing with a video monitor shows a crisp and clear screen. If the screen is not free from stripes then the electrolytic capacitors that smoothen the +5 and +12 voltages should be replaced.



videomod

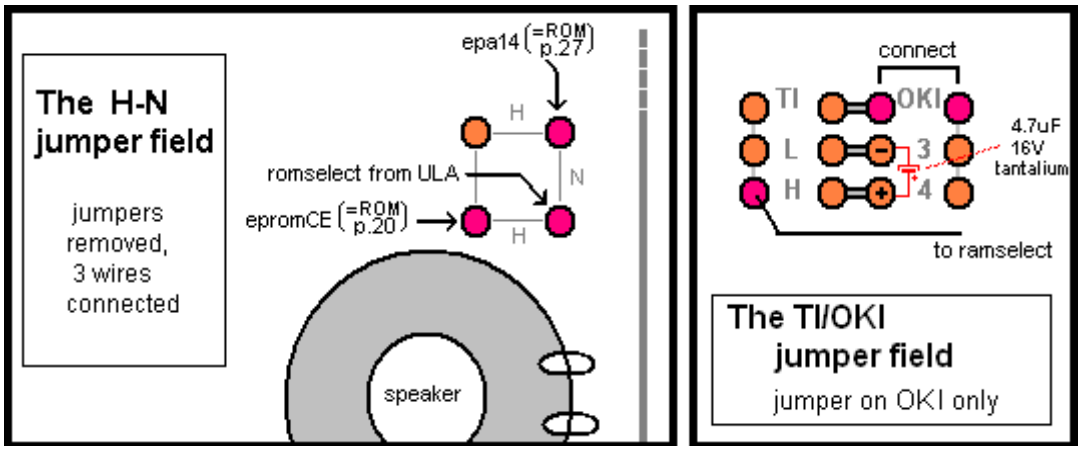
ADDING GAL CHIP

Now I must solder in a pre-programmed GAL chip. Of course such GAL could be socketed, but then a place must be found to hold such socket. And there is little space left inside the rubber 48K for which I prepare this board! Maybe a little PCB should be designed... Anyway, this 2nd prototype can be recognised by the fact that the GAL is glued onto the Z80. This keeps the wiring nicely short. The photo on the next page shows the input signals (red [=grey!] wires) and four outputs (white wires). 'clk' and 'ramselect' are connected with a very thin copper wire. The CF Card is not yet connected.



GALwiring

All twelve signals going into the GAL come from the Z80, except one (romselect) which is taken from the H-N jumper field. From the seven outgoing signals two go to the H-N jumper field, one to the EPROM socket, and one to the TI/OKI jumper field. Also two wires go to a CompactFlash Card connector (discribed further on). Two GAL pins are not connected when using a 64K EPROM. (EP-WriteEnable and EP-A16)
The existing jumpers in the H-N field (on board near the loudspeaker)are removed in the process, and the same goes for any jumper in the TI/OKI field, except that the jumper for OKI remains if it is present, or soldered in when missing. The outside pin of 'H' is connected with 'ramselect' on the GAL.



jumpers

SOME TESTING

Now some simple but serious testing is done with the original ROM still in its socket. After powering up the normal startup message (1982 Sinclair Research) should appear.

PRINT IN 224 should do nothing.
PRINT IN 225 should result in a crash with vertical stripes on screen. You have just seen how the ROM disappeared from (electrical) existance. Restart after disconnecting the power.

Now type:
```
10 CLEAR 32768
20 POKE 40000,111
30 PRINT IN 118, PEEK 40000   (should be 111)
40 PRINT IN 119, PEEK 40000   (should be 0 or some other value)
50 GOTO 30
```
Running this program demonstrates the RAM bank switching.

Now the 16K ROM can be replaced with a pre-programmed 64K EPROM. (How a 128K EEPROM can be fitted is explained further on.)
After powering up a menu should appear, something like the one that is depicted on page 1.

Choose '1', then type and run:
```
10 SOUND 111
20 SOUND 112
30 GOTO 10
```
The new commands can be typed in as ASCII characters when they are preceeded with a space, which will make the editor go from 'K'-mode into 'L'-mode. Tokenizing will take place after entering the line. The keyword SOUND is found in 'R'-mode (which is invoked by pressing keys CAPS+4 simultaniously), under key 'Z'.
This simple program should generate a sound that is more or less like the horn of a police car.

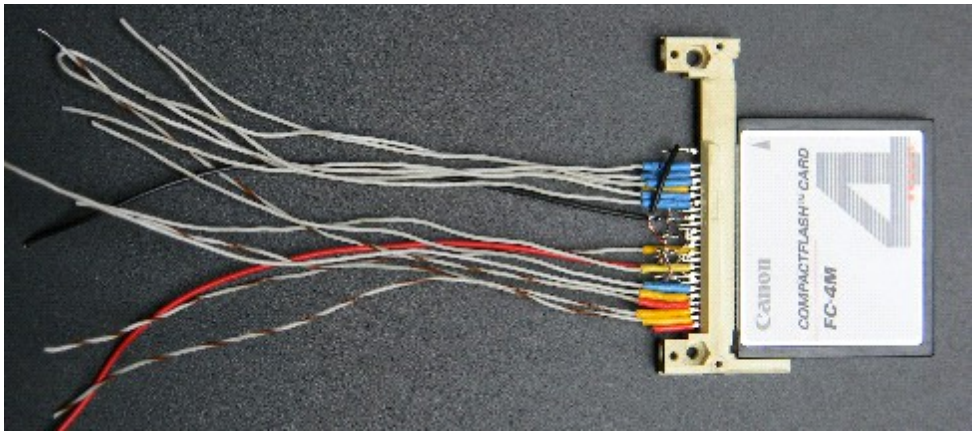
Entering 'PRINT USR 8' should result in printing a number in the range 4.01 to 4.99.
Entering 'CAT 3' should result in the "44 Invalid drive number" message.
Pressing 'Break' (Caps Shift + Space) while editing (entering BASIC lines or direct commands) should bring the last entered direct command back to the lower screen (i.c. in the editor).
Pressing Symbol Shift + Space simultaneously must result in a '$' cursor (the MACRO mode), and a default macro is then found under key "1", ready to be entered. (FOR f=23755 TO ..etc..)

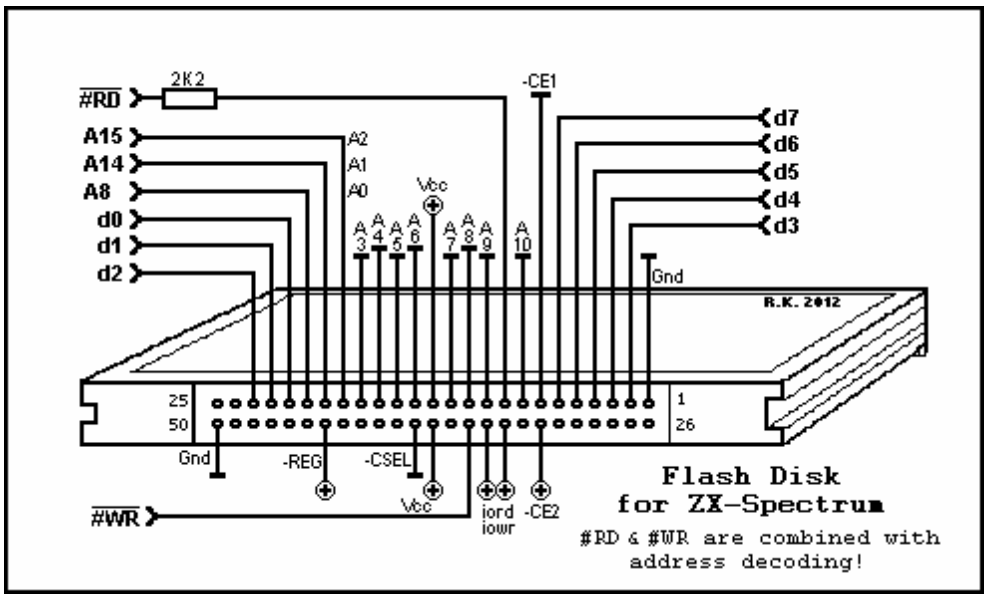
This space is reserved for remarks that do not fit in the text.

*
*
*
*
*
*
*
*
*
*
***

ADDING A COMPACT FLASH CARD

Now a CF connector that already is prepared with wires will be soldered into the system. On the picture is a partly demolished PCMCIA connector shown which is fine for a card that is ment to reside (permanent) internally. (As my plans are.) The wiring is rather simple: +5V and ground, 8 data lines plus 3 address lines go to the Z80, and the #Rd and #Wr signals are connected to the GAL. CF cards can work in different modes (modi) which are choosen by the wiring of certain pins. The wiring used here is for the so called 'memory mode' which allows 8-bit devices to use the full IDE-like capabilities of the card in a direct manner. This memory mode has nothing to do with memory mapping as known in the world of Spectrum and Z80, in fact the CF is just addressed as if it were a peripheral chip like a PIO. The wiring for this 'memory mode' differs from the 'ide mode' as is wired on most IDE-CF adaptors. Such adaptors can only be used in connection with a full IDE interface.

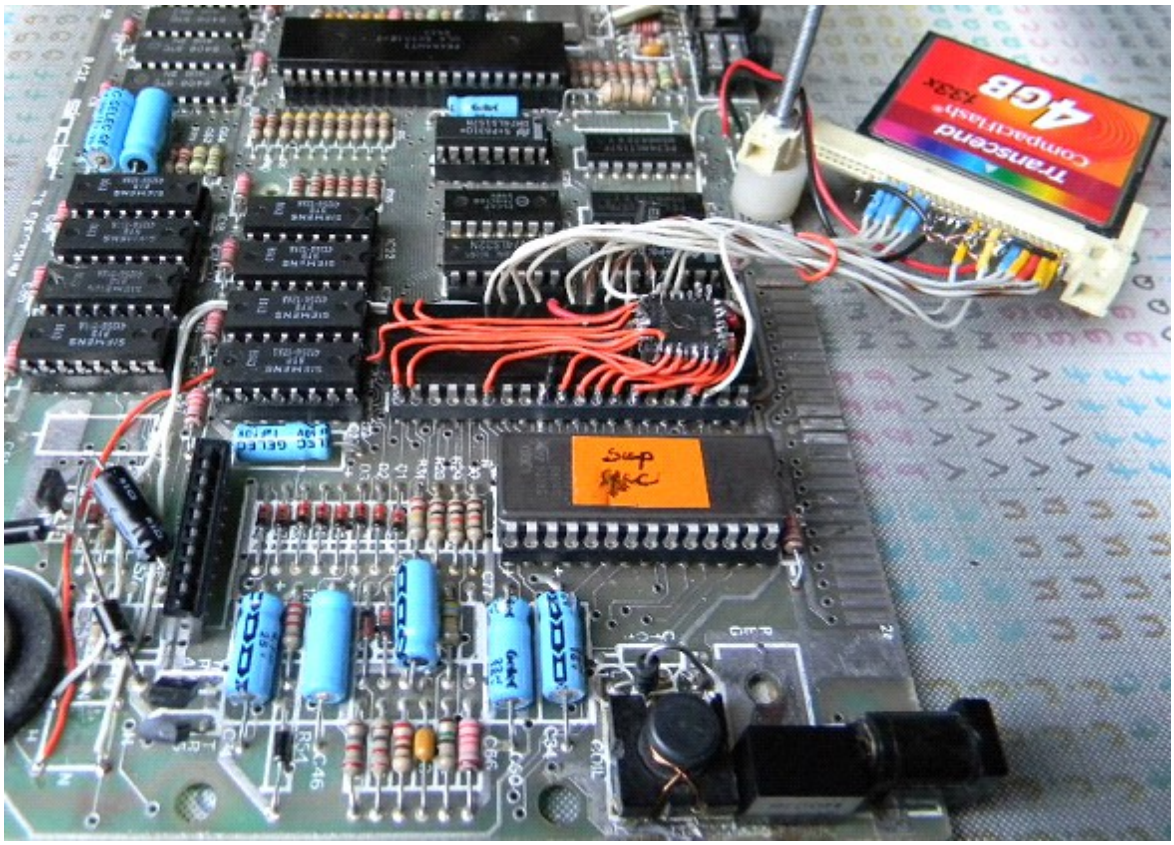




Due to a difference between the timings of the Z80 signals at one side and the needs of a CF card on the other side, the card is enabled all the time (by -CE1) and only controlled by decoded Read and Write signals. The #RD pulse is here fed over a resistor. The wiring used here seems to be over the maximum possible length and not all my cards worked reliably. Inserting an old fashioned resistor (spiral carbon) in the #RD line adds some inductance and solved this problem. A much shorter wiring would be a far better solution!

Because there is no real need to ever exchange a 4Gb CF card, and because I have not yet made a decision about making extra holes in the housing, I just tie the CF card onto the board, or onto the chips or where ever. If IC23, IC25 and IC26 had not been on sockets then the six small chips between ULA and Z80 would have formed a nice platform to tie the CF Card onto. I already mentioned the length of the wires!

On second thoughts I have decided that prototype-2 will stay outside its housing for a while, and serve demonstration purposes.
As shown on this picture:



ready

In the picture above the white wire going to the TI/OKI jumper field can be seen, and also the two white wires plus a red one going to the H-N jumper field. In the picture there is no resistor in the #RD line. The CF card gets its 0V and 5V from the TI/OKI jumper field by means of a red and a black wire. At the same traces a 4.7uF, 16V, tantalium capacitor is placed between +5 and 0V for decoupling purposes. In the picture this small capacitor is hidden behind the white nylon spacer.

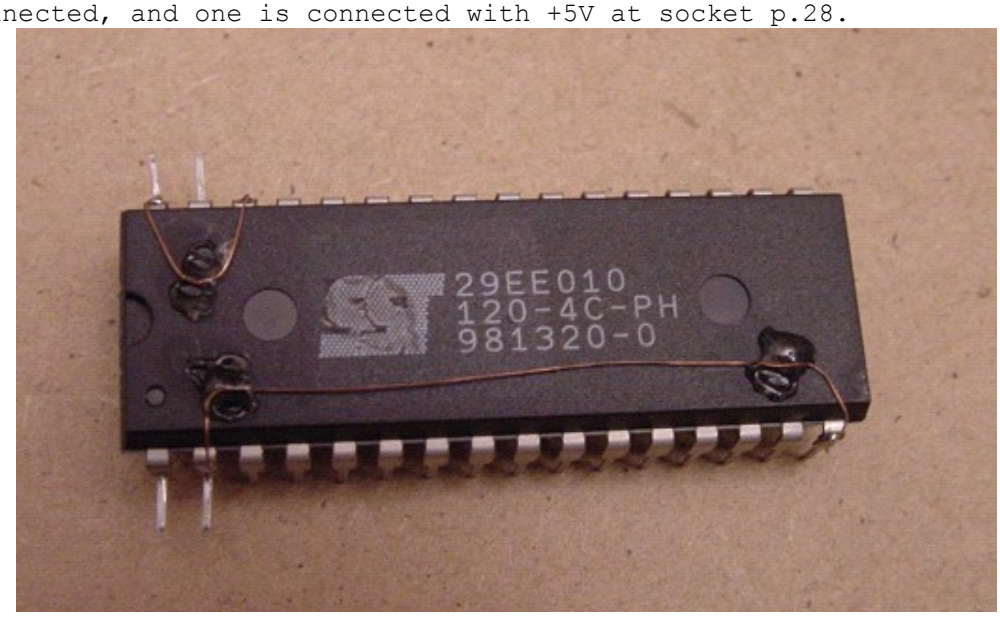
Time for testing again.

Without CF card inserted:
Entering 'CAT 1' should result in "48 Sector error". In fact this is the 'time out' error that occurs when a CF Card is not responding.

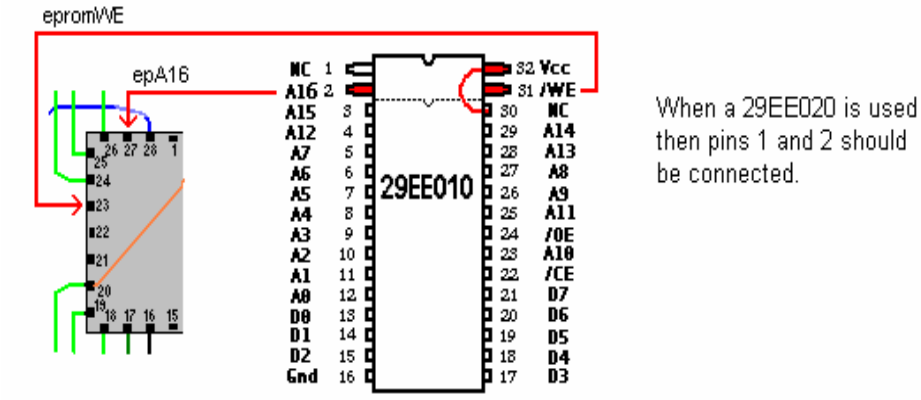
With CF card inserted:
Entering 'CAT 1' or 'CAT 2' should return "45 No format found".
After 'FORMAT 1;"ROOT"' and 'CAT 1' an empty catalogue should appear, with "3580Kb Free".

USING FLASHRAM INSTEAD OF EPROM

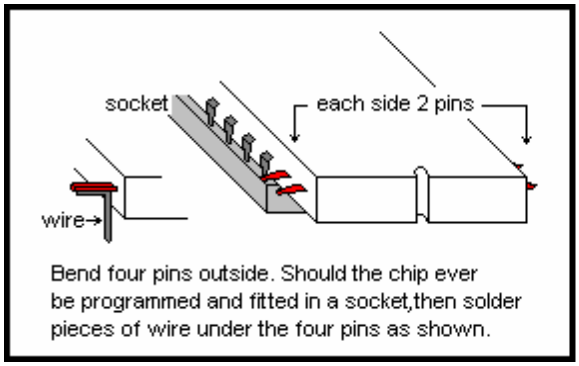
It might be wise to program a FlashRAM with the ROM files before installing it in the Spectrum. This is the simplest way to get the flash programmer code into memory, because a working command interpreter is needed for 'in situ' flashing. The common FlashRAM types 29EE010 and 29EE020 have 4 pins more than the EPROM type that is used in the Spectrum. When 4 pins are left out of the socket, these FlashRams still can be inserted. These 4 pins should be bent sideways. Two of these pins are wired from the GAL chip, one is not connected, and one is connected with +5V at socket p.28.



In the picture above the A16 line is not connected with the GAL but direct with ground, and +5V is taken (over a Not Connected pin of the 29EE010) from the Eprom socket.
Below can be seen how the extra signals are connected to the GAL.



When a 29EE020 is used then pins 1 and 2 should be connected.



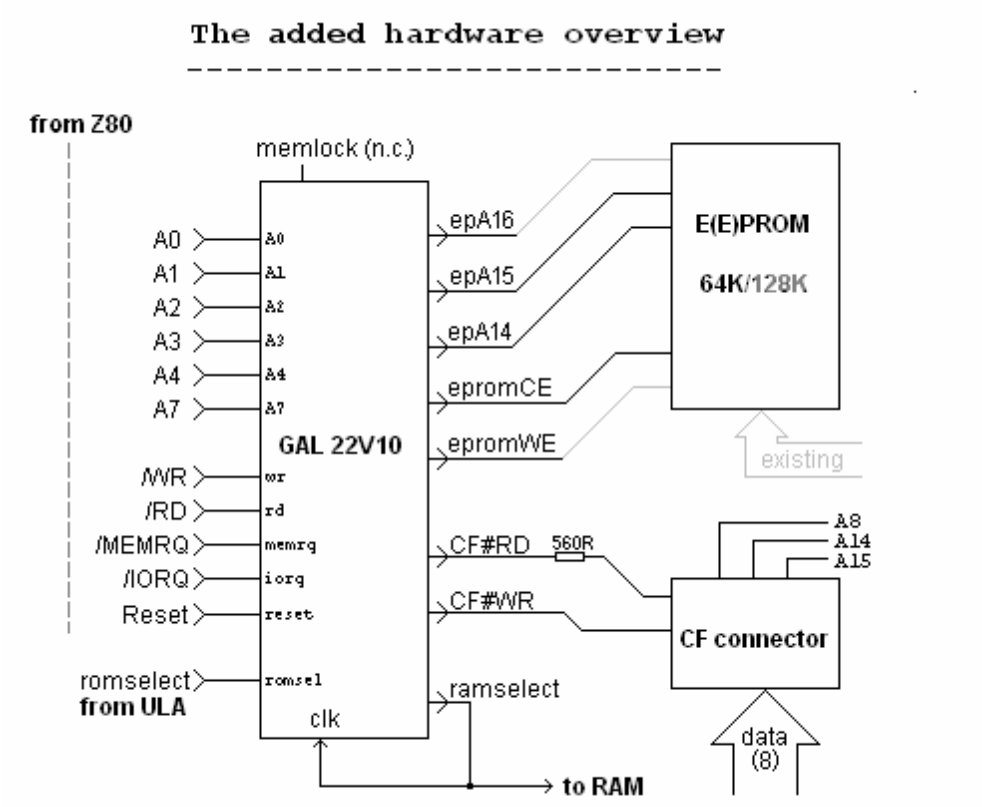
Should it ever be necessary to take the FlashRAM out for re- programming then the four extra pins should not be bent back into position, but 'new' pins should be created by soldering small wires.

FLASHING

ROM banks are electrically switched by instructions IN 224 to IN 231. For that reason I call them "bank 224" and on. Bank 227 (holding the original SpectrumROM) holds a small M.C. routine for flashing an EE- or C type flashRAM chips. With the command PRINT USR 15000 this routine is moved into the printerbuffer. Normally when the original ROM is chosen then bankswitching (paging ROMs) is locked and writing to FlashRAM is disabled. For actual flashing the writing must be enabled and the bank switch must be unlocked.
The method for getting this unlocking done exists of plugging in the power while key "S" is pressed. From there on:

```
PRINT USR 15000 : REM move routine to addr 23300 in printer buff.
PRINT PEEK 23300: REM just a check if routine is present, must be 24
CLEAR 32760
LOAD "romcode" CODE 32768,16384 : REM the code to be flashed must
                                      come from via EAR!
POKE 23302,0 : POKE 23303,128 : REM start addr. of file (32768)
POKE 23304,0 : POKE 23305,64 : REM length (16384)
POKE 23306,0 : POKE 23307,0 : REM first addr to flash (0)
POKE 23308,bank : REM rom bank nr. to flash (224, 225, 226 etc.)
POKE 23309,227 : REM ROM bank to return to when finished
PRINT USR 23300 : REM do it. Any other result as '0' signals an
error. Flashing 16K takes a half second.
```

The both 'BASIC' ROMs (225 and 226) are heavily interacting, so lots of code cannot be changed without a thorough knowledge of the system.
ROM 224 holds the DOS plus the data for the coldstart menu and is interacting with ROM 225.
The 4 ROM pages 228, 229, 230 and 231 are still empty.



```
;#####################################################################
; GAL EQUATIONS for superspec  09-06-2012
;#####################################################################

chip Sup22v10b GAL22V10

;------------------------------------ pins nrs. for DIP24 only! --
clk=1 A7=2 A4=3 A3=4 A2=5 A1=6 A0=7 reset=8 wr=9 rd=10 iorq=11
GND=12 memrq=13 romsel=14 cfread=15 cfwrite=16 ramsel=17
memlock=18 epromwe=19 epromce=20 epA14=21 epA15=22 epA16=23 VCC=24

@ues 4346535045434359    ;CFSPECCY (signature)

equations

epA14 = A0 * /A3 * A7 * /iorq * /rd * /memlock * reset
          + epA14 * reset * A3
          + epA14 * reset * /A7
          + epA14 * reset * iorq
          + epA14 * reset * rd
          + epA14 * reset * memlock

epA15 = A1 * /A3 * A7 * /iorq * /rd * /memlock * reset
          + epA15 * reset * A3
          + epA15 * reset * /A7
          + epA15 * reset * iorq
          + epA15 * reset * rd
          + epA15 * reset * memlock

epA16 = A2 * /A3 * A7 * /iorq * /rd * /memlock * reset
          + epA16 * reset * A3
          + epA16 * reset * /A7
          + epA16 * reset * iorq
          + epA16 * reset * rd
          + epA16 * reset * memlock

ramsel = A0 * /A3 * /A7 * /iorq * /rd * /memlock * reset
          + ramsel * reset * A3
          + ramsel * reset * A7
          + ramsel * reset * iorq
          + ramsel * reset * rd
          + ramsel * reset * memlock

;-----------------------------------------------
memlock = reset * memlock
         + reset * /A3 * A7 * A4 * /iorq * /rd
;-----------------------------------------------
/epromce = /memrq * /romsel
;-------------------------------------------------
/cfread  = /epA15 * /epA14 * /iorq * /rd * /A7 * A3
/cfwrite = /epA15 * /epA14 * /iorq * /wr * /A7 * A3
;-------------------------------------------------

;the eprom-WE signal comes from a flipflop
;the clock of this flipflop ('clk') is connected extern to ramsel!
;-----------------------------------------------------------------
epromwe := reset                          ;only low after power-on
epromwe.oe = /wr * /epromce * /memlock    ;oe follows /wr
;-----------------------------------------------------------------
```

Heavy stuff about the equations for GAL 22V10.

Outputs '**epA14'**, '**epA15'**, and '**epA16'** are data latches that latch the levels on resp. address lines A0, A1, A2. The first term holds the exact conditions for 'true' (making the output high) while all consequetive terms together create the 'not true' conditions for resetting the output. In fact the decoded situations are IN 224 to IN 231.
The output '**memlock**' is a data latch too, but without possibility to reset the output after it is set. It acts on A4.
Output '**ramsel**' is a latch that is constructed identical with epA14 etc., but with different addresses for 'lo' and 'hi' namely IN 118 and IN 119.
'**EpromCE**' just ORs the signals MEMRQ and the existing 'romsel', a job that normally is done inside the Sinclair specific ROM chip.
'**CFread/write**' are OR gates that decode IO instructions on address 127. (The CF card also uses combinations with A8, A14, A15.) The CF Card only 'exists' when the ROM-bank holding the DOS is current, for safety reasons.
The output '**EpromWE**' is a real Flipflop, for which pin 1 of the GAL (when 24DIP, for PLCC it is p.2) is used as clock input. This pin is automaticly assigned to this duty when a registered output is chosen. This 'WRITE-ENABLE' flipflop is set to the Spectrum's /RESET signal, which is low for a moment following a coldstart. This short low pulse is the only LOW that passes by! The state of the flipflop is passed on to epromWE pin whenever epromWE.OE is made valid, in casu when a write to the ROM is made. On other moments the output of the flipflop is just 'tristate' (=hi).
As soon as a (any!) RAMSEL signal reaches 'CLK', then the FlipFlop is set to the current level of RESET, which is HIGH. In this new situation an epromWE.OE signal has no longer any effect on the output level of the FlipFlop, which is now high all the time. So writing to EPROM (FlashRAM) is prevented. This RAMSEL pulse is generated by the default BOOT routine so this routine must be circumvented when ROM has to be flashed. This is done by holding key 'S' held down during a coldstart.
The '**epromWE.OE**' input (which enables the '**EpromWE**' output) detects 'write to ROM' situations.

Heavy stuff about the ROMs

Both hardware and software are built around the idea that it is not necessary to have different parts of an operating system in physically separated ROM chips, and to controll this by a dedicated piece of hardware. That situation once was dictated by commercial laws, but there is no need whatsoever for hobbyists to follow that rule. Different ROMs can reside in different blocks (banks, pages) of the same physical memory chip (EPROM). And then the normally used hardware controlled switch (acting on address 8) can be replaced by a software controlled switch. Which is a rather simple device from the hardware point of view. Several of these switches fit in one GAL.....

The software routines that do the switching (acting on the MC instruction IN A,(n)) are positioned on addresses 15586 (DOS) and 15600 (ext-Basic) in the Spectrum ROM page. The equivalent routines are in the resp. DOS and EXT pages are in the same position. These small routines handle both departure and arrival. The result of the instruction (in the A-register) is not used.

The EXT ROM is paged in at addresses 19, 6833, 15452 and 15457, and then takes over at addresses 11, 85, 657 and 13027 respectively. For the DOS ROM a more direct paging is used on addresses 11 and 5897 from where DOS takes over right away. For that purpose the instruction CALL 15586 is poked in Spectrum ROM in the latter locations. Note that DOS ROM is not limited to 8K but can use the full 16K.
The different methods reflect the many years that ly between the development of these parts.....

When using a (writable) FlashRAM, then writing to this RAM outside the programming environment must be prevented. Some chips await a full programming sequence after one accidental write happens. And then get lost, which causes the Spectrum to crash. Therefore two bugs in the Spectrum ROM must be repaired. The ' 5 byte floating point dump into ROM addresses' must be diverted towards the keyboard buffer. Plus the scrolling of the top screen line into ROM must be stopped. Three patches are made in Spectrum ROM for this purpose.

Heavy stuff about the DOS

The (large) Compact Flash card is divided in 1000 equally sized 'blocks'. Each block acts (in technical sense) as a (hard) disk. The number of 1000 is chosen for making it possible to have a complete overview of all blocks fitting in memory. Describing each block in 24 bytes will thus result in a file of 24k, this allows a future 'DIR manager' program to work with that file. A chosen limit makes it easier to control things, resulting is less overhead. This is important because a nice general speed and a good 'feel' of the system was a design goal.
The blocks are called 'DIR's here. Each of the DIRs carries data that allow a hierarchical system of filing. The DIRs have no number but a name, and are accessed via the 'parent' they are a 'child' of. The hierarchical system is created by using the FORMAT!"name" command.
Files are stored in these DIRs in a consequetive manner, occupying a closed line of storage cells. Deleting files may create gaps between the remaining files. Such spaces can only be reused by new files that fit there. By choosing the size of a storage medium much larger that the total size of the files to store, the need for defragmentation of the gaps is minimised.
The size of DIRs is (loosely) determined on 4 MB. The maximum number of files that can be stored in a DIR is limited to 256, as this is what the existing File Manager program can handle. Practice showed that having a larger amount of files in one DIR is impractical.
On new CFs all storage cells are filled with FF, and therefore are DIRs that are unused or deleted (from the hierarchical system) recognised by having FFFFFF in the first positions.
There always has to be a 'ROOT' DIR to start from. This is the only directory created with the normal FORMAT command, note that this command should NOT be used for making other DIRs as this would destroy the hierarchical system!
At this moment the DOS still uses a (512 bytes) buffer in RAM while loading and saving. That buffer collides with very long programs. So several mega games cannot be not loaded yet. Changing this is on my to-do list, but I do not want to give up the versatile Streams and Channels system introduced by Opus Discovery. So there is another job waiting.
I have been focussed on 'creating' most of the time, and hardly found time for 'using'. In fact building this prototype 2 is the first time for me to play with things. I have already noticed a few issues that I, as a user, want to see changed. For instance the sub directory system seems much too powerfull to be handled by direct commands.....

The current 64K ROM file can be downloaded from: "www.biehold.nl/roelof/supr190712.bin". Those who want a commented assembly source, or need assistance with acquiring or programming an EPROM, can send me a Personal Mail on the WorldOfSpectrum forum.

Added Commands and Functions (BASIC)
==========

Two extra input modes (**F** & **R**) are created, so every key can generate two more keywords. These are visible on the keyboard overview, depicted somewhere in these pages.
Also two other new input modes must be mentioned, ' **$** ' and ' **X** '.

- The ' **R** ' mode is entered by pressing 'INV. VIDEO' (Caps + "4"), and gives access to 34 new commands.
- The ' **F** ' mode is entered by pressing 'TRUE VIDEO' (Caps + "3"), and gives access to 26 new functions. (No functions under the number keys.)
- Embedded True and Inverse **Video** markers can still be entered by means of ' **R** ' mode **1** or **0**.
- The ' **$** ' mode is entered by pressing SymbShift and Space simultaneously, and allows to recall a saved Macro expression created by means of the **MACRO** command. See there for further explanation.
- The ' **X** ' mode is entered by pressing SymbShift and Enter simultaneously, and acts as a forced ' **K** ' mode while the Spectrum actually is in ' **L** ' mode. This allows the use of tokens (keywords) in situations where they normally only can be obtained using tricks. (Like in REM statements)
- As usual, changing ' **K** ' mode into ' **L** ' mode is done by typing a space. This is needed when keywords are entered in single-key ASCII characters instead of (the normal way) by tokens.
- Indeed, all keywords can be entered by just typing in ASCII (' **L** ' mode). Tokenising will follow after entering. (note that 'cat$' has a problem there....)
- Because PROCedures are allowed, misspelled keywords will be treated as if they where procedure names!
- Existing and new keywords can be entered in single-keys without a space, like: DEFFN, OPEN#, CLOSE#, GOTO, GOSUB, DEFPROC, ENDPROC, DEFKEY, DEFWIN, EXITIF.

Added functions Examples

BAND	;binary AND.	PRINT BAND (4,10)
BOR	;binary OR.	PRINT BOR (4,10)
XOR	;binary XOR.	PRINT XOR (4,10)
BIN$	;create a set of binary digits in a string.	LET n$=BIN$ 223
CHAR$	;translate a number into ASCII string .	PRINT CHAR$ 31354
NUMB	;translate string into number.	PRINT NUMB "zz"
DEC	;translate a string with HEX into decimal number.	LET x= DEC "A0B2"
HEX$	;translate a decimal number into a HEX$.	LET n$= HEX$ "FFFF"
DPEEK	;PEEK a 16 bit nr. from a double address	LET basic= DPEEK 23635
INARRAY	;search in 2 dimensional array for $	PRINT INARRAY (z$(n),"pp")
INSTRING	;search n$ inside m$, starting from position n.	PRINT INSTRING (n, m$,n$)
LENGTH	;find mem addr of $ (0), or lenght of dimension (1)	LET n=LENGTH (0,n$)
MAKE$	;create string of given length	PRINT MAKE$ (300,"ha")
MEM$	;move block of memory into string.	LET n$=MEM$ (100 TO 1300)
MOD	;modulus. (get remainder)	PRINT MOD (44,9)
NXTDATA	;preview the type of the nxt data to be read.	IF NXTDATA =2 THEN READ a$
RNDM	;random number inside given range	PRINT RNDM 100
SCRN$	;move block of screen into string	LET n$= SCRN$ (100,100,3,4;1)
SHIFT$	;make changes (1-6 types) in string	LET m$= SHIFT$ (2,n$)
USING$	;format a given number according to caliper	PRINT USING$ ("##.#",12.3456)
WINMENU	;print menu bar in window 'n' and fetch the choice	LET a= WINMENU 1
ERRNR$	;move line+stmt with error into $ (see ONERROR)	LET lino= NUMB ERRNR$(TO 2)
SYSTEM	;=temporary tool =PEEK in alternative RAM bank	
CAT$	;move the result of a 'CAT n' or 'CAT n!' command into a string	
EOF#	;End_Of_File test. EOF# n is valid (= 1) if end of file in stream n is reached ('W'=CASE=unconnected)	

Added Commands Examples

EDIT	;move line or string to edit screen	EDIT n : EDIT n$
PROC	;execute a procedure, using given parameters	PROC sketch x,y
DEF PROC	;start of defining a procedure	DEFROC sketch; x, y
END PROC	;end of defined procedure	ENDPROC
DEFAULT	;assign a value to undeclared variables	DEFAULT a=1, b=2, z=26
LOCAL	;Make variables inside procedures just local	LOCAL n: LET n=999
REF	;Search a program for references	REF "a" : REF a$: REF (a$)
ONERROR	;set Onerr flag, define command as subroutine	ONERROR errnr: PRINT: RETURN
ALTER	;Change screen attributes or expressions	ALTER n$ TO m$ (INK 7 TO INK 1)
AFTER	;after 'n' NMI's a next statement is processed (sub!)	AFTER 100: SOUND 55:RETURN
EVERY	;every 'n' NMI's a next statement is processed	EVERY 100: SOUND 94:RETURN
DO	;start a loop ending with LOOP	DO: PRINT "hallo": LOOP
UNTIL	;condition for both DO and LOOP	DO UNTIL a>3: LET a=a+1: LOOP
WHILE	;condition for both DO and LOOP	DO WHILE a>4: LET a=a+1: LOOP
LOOP	;end of a DO-loop	DO: LET a=a+1: LOOP UNTIL a>3
EXIT IF	;exit from a DO-loop	EXIT IF a=6
ELSE	;exit from invalid IF statement	IF a=6 THEN PRINT 6: ELSE CLS
ELLIPSE	;draw ellipse using xy-center, height, width	ELLIPSE x,y,h,w
FILL	;Fill a closed figure solid or with pattern	FILL INK 3; x, y, address
GET	;Fetch key press without input on bottom line	GET a : GET a$
GET INPUT	;Input in full screen, n=size of field	GET INPUT n; AT 10,10; a (a$)
JOIN	;Add next line to current line or add strings	JOIN : JOIN n$ TO m$
KEYIN	;enter a command or line in BASIC from variable	KEYIN "10 PRINT 1"
MACRO	;User defined keys. With or without inserting	MACRO "2"; "PRINT "" hallo"" :"
ON	;Choose a statement, based on value	ON a: PRINT 1::PRINT 3: GOTO n
DPOKE	;Poke into 2 addresses	DPOKE 16384,65535
POP	;Remove last entry from GOSUB stack	POP : POP n
ROLL	;Roll current window. direction=5678 equ. cursor	ROLL 6 : ROLL windownr, dir
SCROLL	;See Roll. Default is 1 pixel row,	SCROLL windownr, dir, rows
SORT	;Sort strings and arrays	SORT INVERSE n$(2 TO 20)
DEFWIN	;Define window n, topleft2,H,W, Csize, Font	DEFWIN n,x,y,h,w,a,b
WINDOW	;make 0-15 windows 'current', perm. or temp.	WINDOW n :PRINT WINDOW n;"a"
CSIZE	;character width for current window	PRINT WINDOW 2; CSIZE 7;"hh"

FONT ;font type (bold, italic, etc) for current window FONT n. (0-255, not all readable!)
SOUND ;make sliding sounds SOUND n (0-255)
<> ;SPLIT up a line while editing, only available as direct command.
Move cursor behind a " : " in line, then Symbsh+W

Extended existing Commands

PAUSE ;Allowed without number, is PAUSE 0
LET ;multiple assignments allowed: LET a=1, b=2, c=3
POKE string ;POKE 16384," COPY COPY COPY COPY " : REM use keyword
CLS # ;clear main screen with default B/W values
CLS n[,m] ;CLS windownr [, attribute]
WINDOW ERASE ;remove all window definitions
MACRO ERASE ;remove all defined macro's except key "1"
PRINT TO ;print text to pixel positions in main screen
DRAW TO ;draw to absolute x,y, position in main screen
GOSUB / RET ;changed, might give error when used as direct command
CLEAR DATA n$;deletes specified variable
CLEAR LINE n[,m] ;delete line n, or lines from n to m incl.
ERASE DATA ;deletes part of arrays: ERASE DATA a(1)
LIST DATA ;list variables
LIST FORMAT ;listing with formatted layout
INPUT ;see GET INPUT in new commands
'BREAK' ;pressed during editing it will return the last entered direct comand
'BREAK' ;quits from MC after keeping CapsSh+Space down for 5 seconds.
; (assuming interupts are enabled and system variables are intact.)

DOS REFERENCE GUIDE

****

The Disk Operating System is based on disk-images, which images (called 'DIRs') are managed in a hidden hierarchical scheme that makes them appear as sub-directories.
IMPORTANT: For keeping that hierarchical structure intact only the ROOT directory should be created with the command 'FORMAT 1;'ROOT'. Other 'DIRs' should ONLY be created using the new command 'FORMAT!"dirname"'.
The created 'DIRs' appear in a CAT as files that can be loaded. By loading such a DIR it becomes 'current', and will display its contents following a CAT command. The command 'POINT^' will make the 'parent' directory current. DIRs can also be deleted as if they were files, but should be emptied before doing so.
Two different drives (numbers) are provided, mainly for copying purposes. After coldstart both drives are connected to the ROOT directory. The last used drive is the current one, and (important) DOS commands that do not use a drive number will act on this 'current' one.
The whole concept is brand new and some commands that were only ment as tools for experimenting are not yet removed. Use with care.

File names (with a length up to 10 characters) can carry an extension of 3 characters long, following a dot. Such extension must be entered by the user during SAVE or MOVE.
The recognised extensions are:
"BAS","ARR","$AR","COD","SNA","MDR","SCR","SPC",
"128","FIL","EXE","DIR","CRE","TW3","TOR","RAF"
The extensions can (next to other texts) be used for filtering file names like in the command CAT n;"p*.bas". Filtering will work as expected on file names like in LET n$=CAT$ n;"pr?g*". The filtering uses '**?**' as wildcard character and '*****' as end-of-filter marker.
The extensions are shown/not shown depending on using the CAT, CAT n or CAT! command. For LOADing a file (or OPENing etc.) the extensions play no role.
The special file manager, invoked by 'CAT'[enter], will use the extensions for sorting.
After a coldstart or NEW, it is possible to automatically load and run a program from disk. Only type RUN [ENTER] and the system will load a file called 'run' from drive 1. Of course that file must been saved with the LINE option to make it self starting.

Useful USR routines

USR 14070 - resets the disk system.
USR 8 - returns the version number of the dos software.
USR 4007 - allocates system variable space as if Interface 1 was fitted to the Spectrum.

Notations used below:

<drive>
DOS supports two 'drives' which are numbered 1 and 2. Each directory ('DIR') can be connected to each 'drive'. So programs always can work with 'drive 1'.

<filename>
File names can be up to 10 characters long (e.g. "x123") and can be followed by a dot and an extension of 3 characters long like "utility.bas". Upper/lower case makes a diffence.
A file of which the name starts with CHR$ 0, is not displayed in the catalogue but can still be loaded, opened, erased etc.

<file specification>
The channel plus the drive number must be given along with the file name for pointing out a file. The complete file specification is:
"**m**";<**drive**>;<**filename**>. The "m" with semicolon can be omitted from the specification as "m" (disk) is the default channel.

<stream>
The Spectrum uses streams to transfer data from a source (e.g. keyboard) or to a destination (e.g. screen). Every stream ('**#**') has a unique number between 0 and 15 (integer). Streams #0, #1 and #2 are reserved by the Spectrum system. Stream #3 is set up by the Spectrum for use with the ZX printer. Streams #4 to #15 are available to the user and can be connected to 'channels'.
See under <**channel specifications**>.

<channel specification>
Channels represent input and output devices and are represented by a letter. Both upper or lower case letters can be used in the channel specification. DOS adds a few to the existing "K, "S" and "P" channels. The available channels are:
"**K**" - the keyboard. Also outputs to the lower part of the screen. This channel is connected to streams #0 and #1.
"**S**" - the top part of the screen. (Standard connected to #2)
"**P**" - the ZX printer. (Standard connected to #3)

"**d**" - this channel treats the disk as one single entity and is used by the MOVE command. The full specification is "d";<drive>.
"**m**" - a file on disk. Further information is needed to specify it completely. See <file specification>.
"**#**" - this 'stream' channel allows the opening of one stream to another. The full spec. is "#"; <stream>. For example OPEN# 4;"#";1 'links' stream four to stream one. The quotes and the separator can be omitted: OPEN# 4;#1
"**CAT**" - this channel gives access to the disc catalogue file. The complete channel specification is "CAT"; <drive>. This channel is best opened as a random access file with a record lenght of 16.
"**CODE**"- this channel writes or reads directly to/from memory. POINT can be used to select the memory location to read or write from/to.

DOS Commands

****

CAT <channel spec.><drive>	:obsolete long version!
CAT	:invokes the filemanager
CAT <drive>	:display catalogue of specified drive
CAT <drive>**!**	:display file names incl. extension
CAT <drive>;"filter"	:display filtered file names
CAT <drive>;"filter"**!**	:display filtered names incl. extension
CAT?	:list all occupied DIRs by number
CAT*<drive>	:select drive 1 or 2 as 'current', unseen

CAT #<stream>;<drive>	:send catalogue into specified stream. :The stream must have been previously :opened for output.

CLEAR#	:clear out all open streams
CLEAR# <stream>	:clear out the specified stream

CLOSE# <stream> :close specified stream after emptying
:after closing streams 0,1, 2 or 3, they
:are reopened automatically to their
:default (Spectrum) channels.

ERASE <file spec.> :erase the specified file, no error msge
ERASE <channel spec.><drive><file spec.> :obsolete, with error msge
---
FORMAT "m"; <drive>; "name" :obsolete long version do not use!
FORMAT <drive>; "name" :short version do not use!

FORMAT!"name" :create a DIR inside the hierarchy

INKEY$ #<stream> :read one character from specified stream

INPUT #<stream>; variable1; var2;etc. :This inputs values from the
:spec. stream into the spec. variables.

LOAD* <channel spec.> :load a program from the spec. channel
:extensions: CODE, DATA and SCREEN$.
LOAD n :load BAS or COD file numbered n in CAT
LOAD @n,address :load sector nr. n from curr DIR to addr.

LPRINT# <stream>; variable1, var2,... :Print variables to specified
:stream. Exactly as the **PRINT#** command.

MERGE* <channel spec.> :merge a program from the spec. channel
---
MOVE <stream> TO <channel spec.> :transfers information from
MOVE <stream> TO <stream> :an input channel or stream,
MOVE <channel spec.> TO <stream> :to an output channel or
MOVE <channel spec.> TO <channel spec.> :stream.

This command can a.o. move files around, for example: MOVE 1;"myfile" TO 2;"backup". But also MOVE 1;"myfile" TO #2 will work.
Because the free space in each DIR can get fragmented after lots of operations, this space can be defragmented by MOVE "d";1 TO "d";1. You also might say that the occupied space is compacted. This rearranging is needed when the error 'No Room On Disk' is met while a DIR shows enough free KBs.

OPEN# <stream>; <channel spec.> <access> :associate specified stream with specified channel
<access> can be IN, OUT, EXP, RND or be omitted

POINT n :make DIR number n the current
POINT ^ :make the parent DIR current
POINT "name" :make the DIR named "name" current
POINT <stream>; n :set pointer in spec. strm to nth record

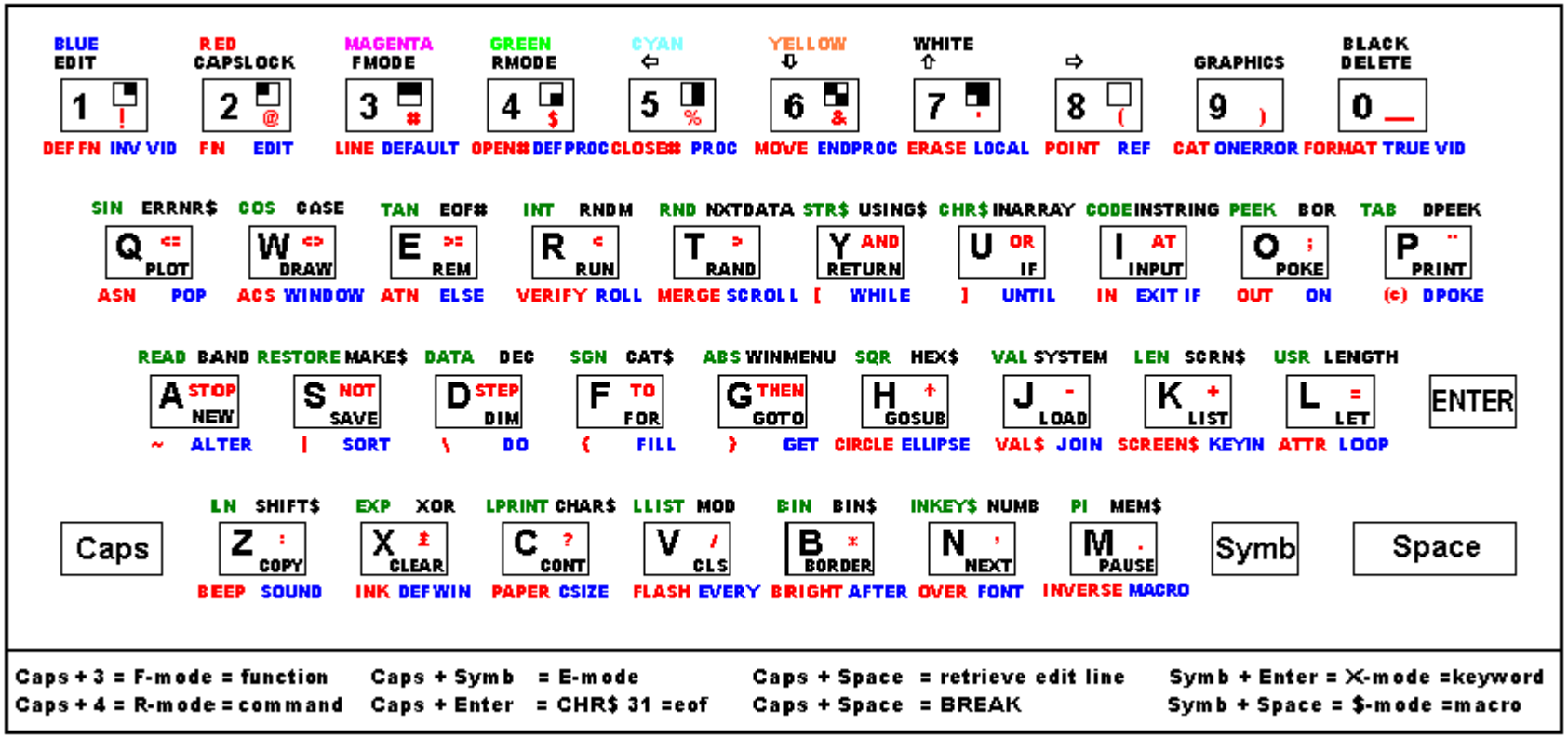
PRINT# <stream>; variable1; var2;...:see LPRINT#

SAVE * <channel spec.> :saves current program
SAVE @n,address :save addr,512 to sector nr. n from curr DIR

VERIFY* <channel spec.> :verify current program

SCREEN$, CODE, DATA, LINE :extension for LOAD, SAVE and VERIFY
---

the end. 2012 Roelof Koning (RoKo).



Caps + 3 = F-mode = function
Caps + 4 = R-mode = command

Caps + Symb = E-mode
Caps + Enter = CHR$ 31 =eof

Caps + Space = retrieve edit line
Caps + Space = BREAK

Symb + Enter = K-mode =keyword
Symb + Space = $-mode =macro

SuperSpec. RK 2011