NASCOM 4 Handbook

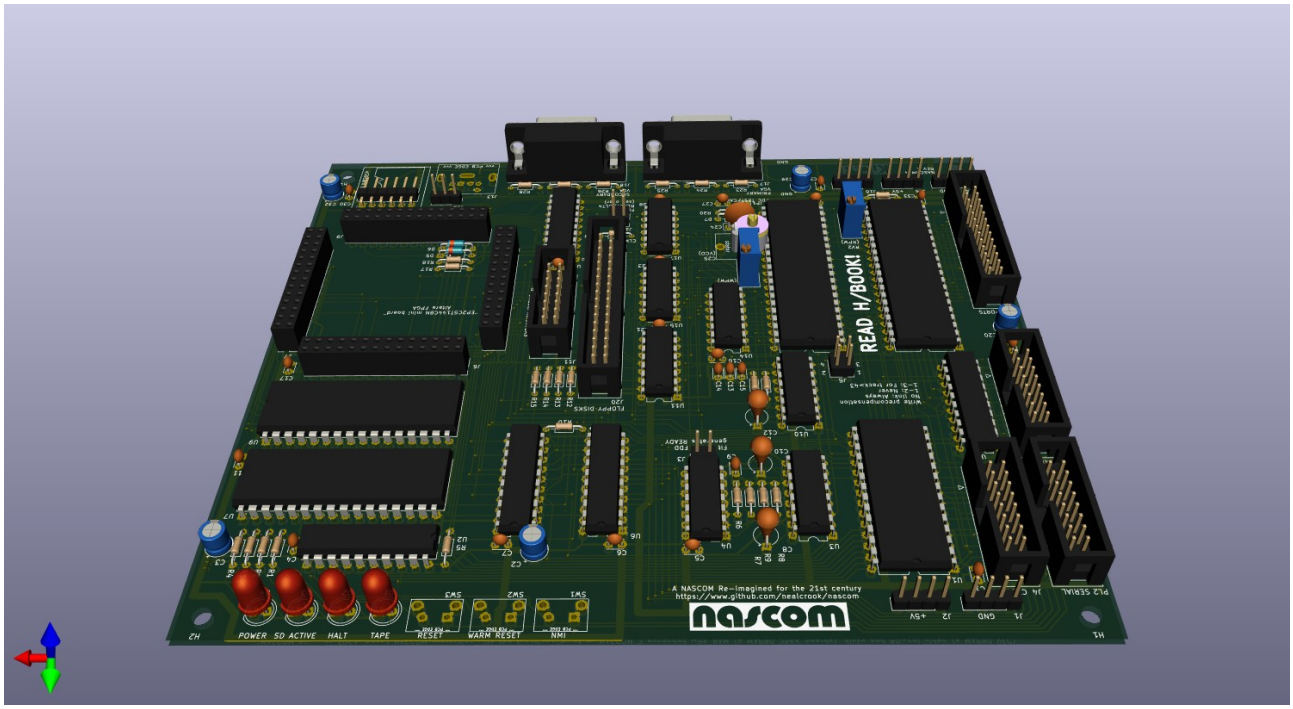


Table of Contents

Introduction....4
Implementation and build options....5
  Minimal system....5
  Expansion options....5
Online resources....6
Errors and omissions....6
Using the push-buttons....6
Board Connections....6
  Power....7
  PS/2 Keyboard....7
  VGA connectors....8
  NASCOM Keyboard connector....8
  SERIAL connector....8
    Connection to a PC/Terminal emulator....9
    Connection to nascom_sdcard....9
  SDcard....10
  PIO....10
  Connection to nascom_sdcard....10
  CTC....11
  Memory Test....11
  Debug Connector....11
  Floppy Disks....11
A guide to the board....12
A guide to the schematics....12
A guide to the FPGA....14
  The I/O bus....14
Programming Guide....16
  Memory Map....16
  I/O Map....18
  SDcard Interface....26
  SBROM Jump-on-reset....27
  SBROM Disable....27
  Reset (cold and warm) and boot....27
  The Special Boot ROM (SBROM)....28
  The Menu System....28
    Menu Examples....29
    Limitations....31
The Video Sub-system....31
  Programmable Character Generator....32
  Programmable Character Generator Example....33
Disk Operating Systems....34
  PolyDos....34
    PolyDos Utilites....34
    SETDRV....34
    CASDSK....35
  CP/M....35

CP/M Utilites....36
SETDRV....36
NASDOS....36
Creating and manipulating SDcard images....37
Construction....39
Testing....46
Stage 1 (Minimal system)....46
Stage 2 (SDcard and external memory)....47
Fault-finding....47
Stage 3 (Serial port)....47
Stage 4 (I/O bus)....47
Stage 5 (NASCOM keyboard)....48
Fault-finding....48
Stage 6 (PIO and/or CTC)....48
Stage 7 (Floppy disk controller) – including calibration....49
FDC Calibration....49
Using 8” disk drives....49
Programming the FPGA....49
FPGA software....50
Reprogramming the EEPROM....50
Shifting configuration data directly into the FPGA....50
Programming service....50
Power supply....51
Mounting the PCB in a case....51
REV A PCB....51
REV A ECOs....52
Acknowledgements....54
Hardware Releases....54
Change History....55

Introduction

This design is an homage to the NASCOM 2 computer produced in the UK between 1979 and 1983. Using a mixture of old and new technologies, it fits all of the hardware capabilities of an expanded NASCOM 2 computer onto a single PCB measuring 170mm x 145mm.

A primary goal of the design is 100% compatiblity with existing NASCOM software, including:

- NASBUG T2, BBUG, NASBUG T4, NAS-SYS 1, NAS-SYS 3
- NASCOM ROM BASIC
- PolyDos
- NAS-DOS
- CP/M (NASCOM and MAP80 systems ports)

The design combines the functionality of these boards:

- NASCOM 2 main board
- MAP80 Systems 256K RAM board
- MAP80 Systems Video Floppy Controller (VFC) board

The design has additional capabilities to allow software control of the system configuration.

The primary connectors/interfaces are:

- LEDs to indicate TAPE and HALT (like the NASCOM 2) and additional LEDs to indicate POWER and SD ACTIVE.
- Push-buttons for RESET, WARM RESET and NMI.
- PIO connector to an actual Z80-PIO chip (like the NASCOM 2)
- KBD connector (like the NASCOM 2) alternatively a PS/2 keyboard can be connected
- SERIAL connector (like the NASCOM 2). The audio cassette interface is not supported, but an alternative mechanism for load from/store to “tape” is provided
- CTC connector to an actual Z80-CTC chip (like the NASCOM I/O board)
- FDD connector to an actual WD2797 disk controller (like the MAP80 VFC)
- 2 VGA connectors. One or two VGA monitors can be attached. The board generates 48x16 video (like the NASCOM 2) and 80x25 video (like the MAP80 VFC) but with the timing adjusted in both cases to drive a multisync VGA monitors and (in the case of the NASCOM video) no “snowstorm” or other video effects. The video sources are generated independently and each can be routed to either VGA connector.
- A micro-SDcard socket. The SDcard holds ROM images that can be loaded automatically at reset. It can also hold disk images for PolyDos and CP/M.

Implementation and build options

The main “guts” of the design are implemented within an ALTERA FPGA. The FPGA is mounted on an off-the-shelf daughtercard which is mechanically and electrically linked to the main PCB through 4 connectors arranged in a square shape.

Minimal system

The FPGA board, connectors, buttons and LEDs (but no other active components) provides most of the capabilities of an unexpanded NASCOM 2:

- T80 CPU from opencores.org – functional equivalent to Z80 CPU, including undocumented Z80 instructions. The core runs at 50MHz but wait states can be inserted (programmable) to allow pseudo-4MHz operation.
- NAS-SYS 3
- 48x16 memory-mapped video (pageable out of the address space), driving VGA output
- (Simultaneous) 80x25 memory-mapped video (pageable out of the address space), driving (same or separate) VGA output
- Programmable character generator (shared between the 48x16 video and the 80x25 video) defaulting to the NAS-CHAR and NAS-GRA character sets
- NMI single-step
- Input through PS/2 keyboard
- 6402 UART (but with Rx buffer and programmable baud rate/estop bits)
- 1 Kbyte RAM (Workspace RAM)

Expansion options

Various expansion paths exist:

- Add a 512Kbyte static RAM to provide the equivalent of 2 MAP80 256K RAM boards; the RAM is paged in a way that is compatible with the MAP80 design.
- Add a micro-SDcard connector and memory card. This allows ROM images to be loaded automatically at reset, allowing access to NASCOM 8K ROM BASIC, ZEAP, alternative ROM monitors (eg NASBUG T4) etc. It also allow the “Local SD” versions of PolyDos and CP/M 2.2 to be run.
- Add a handful of 74-series logic chips for voltage level translation and buffering in order to create a 4MHz I/O expansion bus. Then..
- Add a 74-series logic chip to allow the connection of a genuine NASCOM keyboard (this can operate instead of or in parallel with the PS/2 keyboard).
- Add a Z80-PIO and/or a Z80-CTC for interfacing to external devices or playing with interrupts.

- Add a WD2797 disk controller and associated passives and 74-series parts to allow connection of upto 4 floppy-disk drives (or Gotek simulated drives).

Unlike an emulator, where some other processor is used to imitate the operation of a different system, an FPGA is a true logic implementation, allowing reproducible cycle-accurate timings and true parallelism for the handling of asynchronous activities like serial data transfer and interrupts. The use of contemporary VLSI parts like the Z80-PIO, Z80-CTC and WD2797 ensures a high degree of compatibility with the original designs.

Online resources

This manual and a PDF version of the schematics can be found at https://github.com/nealcrook/nascom in the nascom4 folder.

Additional design information (FPGA code, KiCad schematic/PCB design database, SBROM code) can be found at https://github.com/nealcrook/multicomp6809/tree/master/multicomp/NASCOM4

My other NASCOM-related stuff can be found at https://github.com/nealcrook/nascom.

Errors and omissions

You may find some information here wrong or confusing. You may look on my github repository and be unable to find something that you are looking for: or it may seem wrong or out-of-date. Please report any such instances to the author.

Using the push-buttons

The reset/warm reset buttons can be used like this:

- On powerup or after pressing the reset button, a boot menu will be loaded from the SDcard. Use the keyboard to select a menu option (The Menu System on page 28 describes how the menu works and how it can be customised)
- On powerup or after pressing the reset button, press the warm-reset button to start up NAS-SYS3 (see Minimal system on page 5). This mechanism works whether or not an SDcard is fitted.
- After booting the system using either of the previous two methods, pressing the warm-reset button performs a hardware reset but maintains the exisiting configuration.

The NMI button can be used to generate a non-maskable interrupt. On all NASCOM monitors this is used by the single-step logic and so the effect is to interrupt program execution, display the register contents and return to the monitor command-line. If NAS-DEBUG has been initialised, an extended register dump will be displayed.

Board Connections

This section describes power hookup and devices that can be connected to the NASCOM 4 board.

Power

The board requires a regulated +5V supply. It also uses +3V3, which is generated on the FPGA daughter-card. The FPGA connectors supply +3V3 and the 0V reference from the FPGA daughter-card to the main PCB but there is no +5V supply on those connectors. Therefore, +5V power must be wired to the FPGA daughter-card *and* to the main PCB.

The simplest way to do this is to use a 5.5mm diameter barrel connector to supply power to the FPGA daughter-card (cente/pin is positive; check the polarity carefully) and to solder a flying lead from the FPGA daugher-card to the main PCB (eg to J2 or J15 on a REV A PCB or, more conveniently, to J100 on a REV B PCB).

An alternative would be to wire power to the main PCB (to J2/J15 and to J1/J14) and to connect a flying lead from those connectors to a barrel connector to plug in to the FPGA daughter-card.

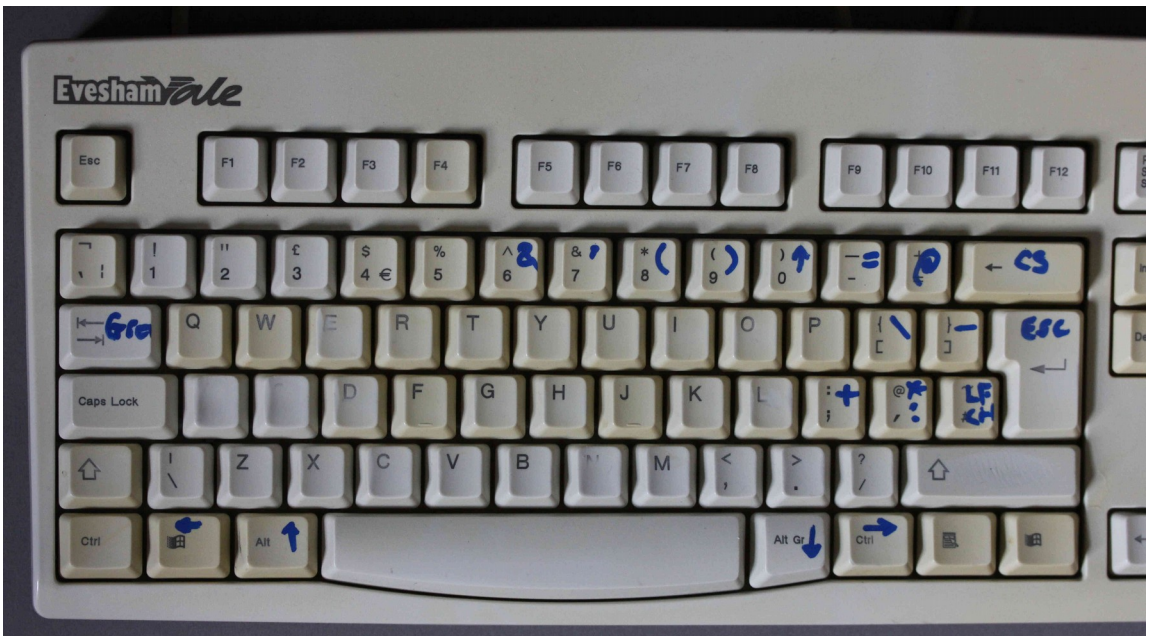
TODO measure and describe power requirements.

PS/2 Keyboard

Yes, I know you'd prefer if it supported a USB keyboard, but that is a much harder interfacing problem. It used to be possible to buy USB to PS/2 adaptors, but they are passive devices, and only work for certain old designs of keyboard that support both protocols. If you don't have a PS/2 keyboard in a cupboard somewhere, try a local charity shop or freecycle/freegle group or check on EBAY.

All PS/2 keyboards should work happily on 5V power. Some will work on 3V3. Set jumper J12 appropriately. If you run at 5V you must fit Zener diodes D5, D6 to avoid blowing up the inputs of the FPGA. It's probably safest to always fit them.

Some of the keys are unused (do nothing), and some need relabelling to reflect their new function. You can do a tidy job with a fine-tipped paint pen or, like me, a messy job with a not-fine-enough paint pen. The changes are shown in the photo below.



Both r-CTRL and r-WIN should work as right-arrow keys. Also, the 4 cursor keys should work. If you have other suggestions for keys that should be decoded, let me know.

The Caps-lock, Scroll-lock and Num-lock keys toggle their respective LEDs on the keyboard when pressed, but they have no other effect.

VGA connectors

The two VGA connectors should drive any normal LCD. If you find a monitor that does not behave, let me know. In early code releases, the final pixel of the NASCOM screen seems to be truncated. That was fixed in Rev1.1 and later releases.

The NASCOM 48x16 format is implemented using a 800x600 mode at half rate so that the effective clock rate is 25MHz.

The MAP80 VFC 80x25 format is implemented using 640x400 mode.

NASCOM Keyboard connector

The KBD connector, PL3, is electrically and mechanically equivalent to PL3 on the NASCOM 2.

SERIAL connector

The SERIAL connector, PL2, is somewhat equivalent to PL2 on the NASCOM 2; it only has 5V signalling and it does not provide connections for an audio tape. It *does* provide all of the signals for direct connection to my nascom_sdcard board, or to a USB-to-serial adaptor.

After (cold) reset, the default signalling on this interface is 115200baud, 8-bit data, no parity, 1 stop bit. The configuration of the serial interface is controlled by port $1D and is not affected by warm reset.

Connection to a PC/Terminal emulator

Suitable USB-to-serial adaptors are available on ebay, aliexpress etc. An adaptor with 5V or 3V3 signalling will work (some adaptors have a slider switch to select the signalling). Connect the GND, then connect the adaptor's TXD to the NASCOM RXD and the adaptor's RXD to the NASCOM TXD.

When running any of the NASCOM monitor programs, the NASCOM keyboard and the serial port are both polled for input. However, output is sent (by default) only to the memory-mapped VDU. Issue the command "X22" from NAS-SYS to echo output to the serial port. Configure the external terminal (emulator) for 115200baud, 7-bit data, 1 stop bit (7S1).

The serial speed and other communication parameters can be configured by issuing "O 1D" commands from NAS-SYS. Refer to the description of port 1D in section I/O Map on page 18.

Connection to nascom_sdcard

The nascom_sdcard serial interface can be used as a virtual "cassette tape" interface when the NASCOM 4 is running NAS-SYS 1 or NAS-SYS 3 (in the future it may also work with NASBUG T4).

Connect the nascom_sdcard using a 16-pin cable to the NASCOM 4 SERIAL connector, PL2. Press RESET and boot into NAS-SYS 1 or NAS-SYS 3.

Observe that the NASCOM cursor is blinking erratically; that is because nascom_sdcard is pulling the RXD line active, delivering a constant stream of NULL characters which the NASCOM is ignoring.

Issue the following commands to NAS-SYS:

```
O 1D 18
(you might get a spurious character printed here; back-space over it)
R
```

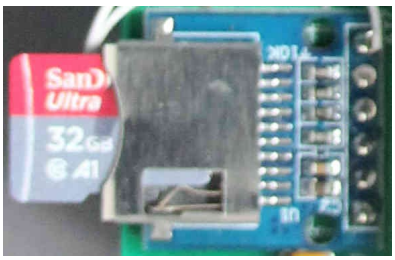
The first command reconfigures the serial interface: it selects external baud clocks (i.e., so they are supplied from nascom_sdcard) and inverts the RXD and TXD signals. The second command makes the NASCOM load the serboot program from nascom_sdcard and execute it. The "DRIVE" LED on the NASCOM and the nascom_sdcard should illuminate for a few seconds and then the screen should look like this:

```
O 1D 18
R
0C80 006E .
E0C80
NAScas>
```

At this point, nascom_sdcard can be used normally (refer to the user manual). After a warm reset, the serial interface will remain configured correctly: type EC80 (if the memory has not been corrupted) type R to reload serboot (or just EC80 if memory has not been corrupted).

SDcard

The interface supports standard capacity (SDSC) and high capacity (SDHC) cards. Physically, a micro-SDcard is required. The card only pushes about half-way into the socket and there is no spring-loaded eject mechansism: simply pull the card out.



PIO

The 26-way PIO connector, PL4, is electrically and mechanically equivalent to PL4 on the NASCOM 2.

Connection to nascom_sdcard

The nascom_sdcard serial interface can be used as a virtual disk interface when the NASCOM 4 is running NAS-SYS 1 or NAS-SYS 3. Currently this supports PolyDos and NAS-DOS.

Connect the nascom_sdcard using a 26-pin cable to the NASCOM 4 PORTS connector, PL4. Press RESET and select the menu option that includes “POLYDOS2-SD”. The “SDACTIVE” LED on the NASCOM should illuminate for a few moments and then the screen should look like this:

```
Boot which drive? _
```

In response, type 0. After a few moments you should see this exciting message:

```
PolyDos 2 (Version 2.1)
Copyright © 1982 by
PolyData microcenter ApS
$
```

At this point, nascom_sdcard can be used normally (refer to the user manual). After a warm reset, the system will execute the PolyDOS ROM automatically, stopping at the “Boot which drive?” message.

TODO add menu entry for NAS-DOS-SD and example here for starting it.

Note: If nothing’s attached to the nascom_sdcard SERIAL connector, the nascom_sdcard firmware might decide that a tape operation is in progress, which will stop the parallel interface from working. The cleanest fix for this is to add a 100K resistor on nascom_sdcard, from J2/3 (“DRIVE”) to J2/6 (“GND”). TODO: test.

CTC

The CTC connector, J4, is not like anything on the NASCOM I/O board. I chose a different pin layout; one that allows the use of 2-pin jumpers to connect the clock to any channel and to daisy-chain from one channel to the next. The connector includes +5V power, ground and the I/O bus 4MHz clock.

Memory Test

One of the menu options is to load “NAS-SYS3 + MEMORY TEST”. This loads David Allday’s MAP80 RAM test program.

Because the menu system loads programs with a granularity of 512 bytes, and the memory test program loads into workspace RAM at $C80, the SBROM stack gets corrupted during the load and does not terminate cleanly. Therefore, you must manually disable the SBROM before running the memory test. From NAS-SYS:

```
O 18 19
E C80 8000 0 8
```

The execute command uses 32Kbyte paging at $8000 to test 8 pages (512Kbytes) of memory. For more details refer to https://github.com/dallday/map80memorytest

Debug Connector

The debug connector, J11, has no function currently. It is connected to all of the unused/unassigned FPGA pins. It might get used for debug, or it might not ever get used at all.

Floppy Disks

The floppy-disk drive connector, J20, should be “standard” but there are some variations on the way that drive select 3 and ready are assigned; I copied the MAP80 VFC jumpering and I hope that this provides sufficient flexibility.

Use a 1-1 cable (not an IBM-PC ‘twist’ cable) to connect floppy disks.

To connect a Pertec FD250, leave J5, J10, J3 open. On the FDD, select drive 0 for the first drive and remember to fit a resistor terminator pack on the last physical drive.

To connect a Gotek FDD emulator, leave J5, J10, J3 open. On the FDD, fit jumper S0 to select it as drive 0. I used Flash Floppy v3.2 to successfully boot PolyDos 2, CP/M 2.2 and CP/M 3. Setup files are available on request.

If you attach any other FDD type, please report back to me the settings for successful operation so that I can include them here.

A guide to the board

The power grids are rather haphazard (I'd do that differently another time; the REV B is soemwhat tidier in this regard) but there are connectors top and bottom with 0V if you want somewhere to earth a scope probe.

I spent a lot of time cramming components close together so take care when selecting parts that they will fit in the space available. I have tried to keep the same orientation for all connectors and for components in general to reduce the chance of errors in assembly or when connecting stuff.

Likewise, I have tried to make the silk-screen as useful as possible, with all of the reference designators visible and details on the link settings added for reference.

Even if you don't put the board in a case, you might consider putting it on stand-offs to avoid shorting out anything on the bottom.

A guide to the schematics

The schematic set covers 6 sheets and should be readable when printed at A4. The schematics and PCB layout were done using the open-source tool KiCad. The whole design database is available, as well as a PDF render of the schematics.

The schematics are hiearchical, with Sheet 1 as the root. The 5 rectangles labelled "Sheet" in the bottom-right of Sheet 1 represent the other 5 sheets of the schematic set. All signal connections between pages use global nets, represented by net names "in a box". Signals that originate on a page and never go off-page use local nets, represented by net names that are not "in a box". For example, on Sheet 2, PIO_B4 (top right) is a local net and CLK4_5V (bottom central) is a global net.

Sheet 1 contains 6 strange components: 4 holes and 2 logos (the NASCOM logo in silk-screen and the KiCad logo in solder-resist). These could be place directly on the PCB without appearing on the schematic. However, they then disappear each time the PCB is updated from the schematic. By creating a component and placing them on the schematic they maintain their existence/position through PCB updates.

The dotted boxes and dimensions on Sheet 1 are informational only.

The 4 FPGA connectors are orientated to match their positions on the PCB and are mirrored relative to the FPGA daughter-card, since that daughter-card is mounted face-down on the PCB.

Sheet 2 contains the external RAM. U9 is required if you want to do anything fun. U7 is optional. U9 provides 512Kbytes of RAM and the only exisiting use for memory above 64K is as a RAM-disk under CP/M. Fitting U9 just allows a bigger RAM disk.

U2 is the data bus buffer for the 4MHz I/O bus. Use of an LVC device means that it can be powered at 3V3 but act as a level translator for the I/O bus, which uses 5V signalling.

U5, 6, 16 are LVC buffers. The original design intent was to buffer all signals between the FPGA and the 5V world, but I ran out of space so my revised rule was (1) Buffer (level translate) any 5V signal that is coming in to the FPGA, (2) Buffer as many FPGA signals going out as possible, prioritising signals with a high fanout.

Sheet 3 contains the Z80-PIO and Z80-CTC and their connectors to the outside world. PL4 is wired identically to the same-named NASCOM connector. The interrupt daisy-chain puts the PIO at the head of the chain, just as the NASCOM2 does.

J4 can be used to connect the CTC to the Real World or it can be used to connect clocks to different CTC channels and to daisy chain from one channel to another; the pinout has been designed so that all of this can be achieved using 2-pin jumpers. Since the main 4MHz clock of the I/O bus is exposed unbuffered on this connector[1], be careful when connecting jumpers.

Sheet 4 and Sheet 6 are the floppy disk controller. This is almost a carbon copy of the MAP80 VFC design. The WD2797 is a nice chip with data separator built in and so this is probably as simple as a disk controller design can get. The buffers driving the disk connector are probably over-engineered: intended to drive 0.5m of ribbon cable, but will probably end up driving 200mm to a GOTEK. There is a solder-pad jumper to put the FDC into test mode, for adjustment of the data separator and the write precompensation circuit (see Stage 7 (Floppy disk controller) – including calibration on page 49).

The monostables on Sheet 6 deserve some explanation (any any corrections to my description are welcome). Port $E4 is the drive-select register. Each write to this port retriggers U3A. This asserts DRIVE_ENABLE to turn on the drive motor; subsequent writes to this port as the disk operation progresses retrigger U3A keeping DRIVE_ENABLE asserted and thus keeping the motor running. At the end of the disk operation, writes to port $E4 cease and the monostable expires, turning off the motor. The *first* write to port $E4 in a disk operation (the write that causes DRIVE_ENABLE to assert) also triggers U3B. This provides a delay between starting the drive motor and the drive appearing “ready” – which is signalled to the FDC by the assertion of READY (READY is also level-translated and provided as an input to the FPGA because its state can be read back in port $E4. The FDC holds-off its operations until it sees READY assert. This idea of using a monostable to generate READY on the controller board was common for early disk drives. Later drives generate READY directly but the pin used for this function was non-standard. Jumper J3 (Sheet 6) controls whether READY is supplied from the monostable or from the disk drive. Jumper J10 (Sheet 4) controls whether READY comes from pin 6 (ISLT3) or pin 34 and allows pin 6 (ISLT3) to be used as drive select 4 output.

Sheet 5 is mostly connectors. The two DB15 connectors provide VGA output. The video output is monochrome, but two pairs of resistor triplets allow a particular colour to be selected. For example, fitting all resistors of a triplet will give a white display whilst just fitting the _GREEN resistor will give a green display. Two 16-pin headers provide the connections for KBD and SERIAL, which match the pinouts of the same-named connectors on the NASCOM 2. The 6-pin connector for the uSDcard breakout board is designed to match the pinout and footprint of a small, passive, adaptor PCB available on AliExpress. This seemed more straightforward than having to solder a surface-mounted socket into position. Pin 3 is the FPGA output, supplying data to the SDcard (sometimes named Master Out, Slave In: MOSI) and pin 5 is the FPGA input, receiving data from the SDcard (sometimes named Master In, Slave Out: MISO). Pullups for the signals are supplied by the adaptor PCB.

1 Maybe that wasn’t a brilliant idea..

The 6-pin mini-DIN is the PS/2 keyboard connector. The PS/2 protocol is similar to I2C and the clock and data signals are open-collector and so are provided with pull-up resistors here (it seems ambiguous whether the keyboard itself provides them). The keyboard runs at 5V and so the signals will swing to +5V if there are pull-ups on the keyboard itself. The Zener diodes are intended to clamp the signal swing to 3V3 to protect the FPGA inputs.

Finally, there is 1 active device on this sheet: an 8-bit buffer. This buffer serves 2 purposes. Its primary purpose is that it is enabled during a port 0 read, allowing keyboard scan data to be read. Its secondary purpose is that it is periondically enabled so that the I/O data bus does not float for an extended period (See The I/O bus on page 14).

Sheet 6 contains monostables for the disk drive interface (see the Sheet 4 description above), +5V decoupling capacitors (3V3 decoupling capacitor are on sheet 2), power connectors and some IC power pin connections.

A guide to the FPGA

TODO hierarchy, overview, simulation TB. For tool-chain see..

http://searle.x10host.com/Multicomp/index.html

The design uses 13Kbytes of internal ROM/RAM (internal to the FPGA, ROM and RAM are equivalent), as follows:

- 2Kbyte ROM for NAS-SYS, 2Kbyte ROM for MAP VFC boot, 1Kbyte ROM for Special Boot ROM
- 4Kbyte ROM for character generator
- 1Kbyte RAM for NASCOM video, 1Kbyte RAM for NASCOM workspace, 2Kbyte RAM for MAP VFC video.

The I/O bus

The PIO, CTC, FDC and keyboard buffer are connected to the I/O bus. When the T80 CPU within the FPGA generates a cycle directed at this bus, the CPU is stalled and the cycle is performed on the bus with timing to mimic a 4MHz Z80. This function is performed by the nasBridge module within the FPGA. One of the design goals for NASCOM 4 was to support daisy-chained vectored interrupts from Z80 peripherals (a PIO and a CTC) connected to this bus. Meeting this goal is one of the most complex parts of the design. The nasBridge module has these features/functions:

- A free-running 4MHz clock (for the PIO and CTC and a free-running 1MHz clock (for the FDC)
- A reset sequence that lasts multiple cycles and that asserts /M1 in order to reset the PIO
- Regular /M1 cycles (asynchrononous to internal T80 operation); the PIO (and CTC?) synchronise the assertion of /INT to /M1. Without /M1 cycles, these devices will never assert /INT. These /M1 cycles need to have /RD driven so that they are not confused with a PIO reset sequence.

- Prevent the tri-state data bus from floating for an extended time. This could be achieved by keeping the data bus buffer, U2, enabled and directed towards the I/O bus. However, that would cause a lot of high-frequency switching on the bus even when nothing is accessing the I/O bus. A more polite solution is achieved by output-enabling the keyboard buffer, U8, periodically, allowing it to drive a non-Z value on the bus. It is important that the buffer never drives the value $4D on the bus, because this could be confused with a RETI (return from interrupt) instruction. If no keyboard is connected, the inputs to U2 should float high (TODO HCT doesn't seem to do this). If a keyboard is connected the MSB is tied high and so it will drive a non-$4D value. Worst-case, it will drive a $ED for multiple cycles but that will not cause a problem for interrupts.
- I/O read and I/O write cycles from the T80 addressing an external device cause the bridge to stall the T80 and propagate the cycle. Because the I/O bus clocks are free-running and /M1 cycles are generated on the bus in a way that's decoupled from T80 cycles, the first step is to wait until any /M1 cycle has completed and the correct alignment with the 4MHz clock is achieved.
- Spy on T80 code execution to detect a RETI (return from interrupt) instruction and propagate it onto the I/O bus so that the PIO and CTC can control their interrupt daisy-chains correctly (see reference below). This is the most fiddly part because RETI is a 2-byte instruction ($ED $4D) and different things happen in the PIO/CTC after each of the two bytes. The simplest approach would be propagate each byte independently when it's seen in an /M1 cycle, but this would be inefficient because it would mean stalling the T80 each time an instruction with the $ED prefix is executed (the only way to drive the $ED value onto the I/O bus is from the FPGA data bus output via the data bus buffer U8. Therefore, the $ED value cannot be driven autonomously from T80 external memory cycles). The solution is to detect the pair of instructions, then stall the T80 on the second one and sequence them both on the I/O bus. This is more complex but most efficient in terms of bus usage.

The nasBridge (only) responds to these I/O port addresses:

- Port $00 read (NASCOM keyboard). The bridge generates a read strobe so that no external decode is required.
- Port $04-$07 read/write (PIO). The bridge generates address lines A[1:0] (buffered from the external memory bus) and a PIO chip-select so that no external address decode is required.
- Port $08-$0B read/write (CTC). The bridge generates address lines A[1:0] (buffered from the external memory bus) and a CTC chip-select so that no external address decode is required.
- Port $E0-$E3 read/write (FDC). The bridge generates address lines A[1:0] (buffered from the external memory bus) and a FDC chip-select so that no external address decode is required.
- Port $E4 write (FDC drive select/enable). The bridge generates a write strobe so that no external decode is required.

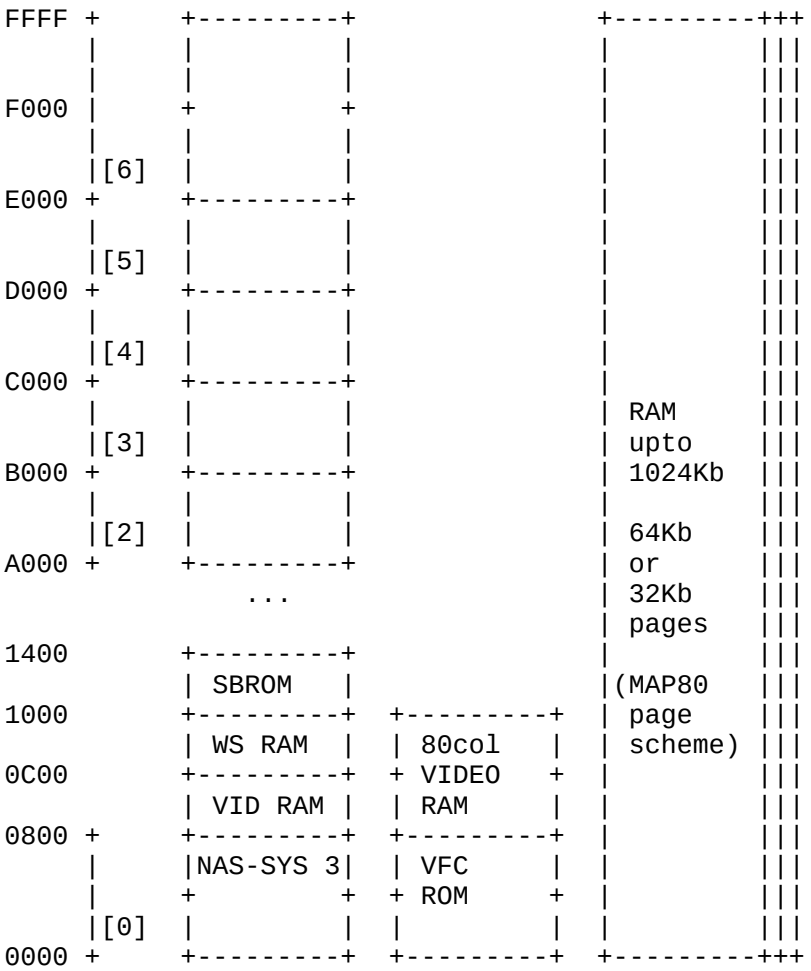
All other I/O port addresses are either decoded within the FPGA or are ignored.

Programming Guide

Memory Map

The Z80 CPU can directly address 64Kbytes. The memory map at reset is shown below. There are four columns in the diagram. The first column (vertical line with 6 sections marked [0], [2], [3], [4], [5], [6]) represents the write-protect regions. The second column with boxes marked “NAS-SYS 3”, etc. represents the (modified) NASCOM address space that exists after reset. The third column, with boxes marked “VFC BOOT ROM”, etc. represents the address space associated with the video part of a MAP80 VFC card and the final column represents the address space associated with upto 4 MAP80 256K RAM cards.

The final column is implemented by 1 or 2 SRAM devices external to the FPGA. All of the other memories and all of the protection logic and memory-mapping control are implemented within the FPGA. The write-protect and the SBROM are new/original features for NASCOM4.



The CPU “views” the 4 columns from left to right. This means that write-protect has the highest priority, and RAM the lowest priority. This is equivalent to the functionality implemented by the NASBUS signal “RAMDIS” and the implications will be described after a block-by-block tour of the memory map.

The write-protection is controlled by register PROTECT (I/O port $19). There are 6 protection regions: a 2Kbyte region at the bottom of the address space (corresponding to the NAS-SYS address space), 4, 4Kbyte regions starting at $A000 (corresponding to EPROM space on the NASCOM 2) and an 8Kbyte region starting at $E000 (corresponding to the BASIC ROM on the NASCOM 2).

The memory mapping of the NASCOM address space is controlled by REMAP (I/O port $18). The NAS-SYS 3 ROM can be disabled. The video RAM can be disabled or remapped to start address $F800 (for NASCOM CP/M). The workspace RAM can be disabled. The Special Boot ROM (SBR) can be disabled (its disable is special, as will be discussed later in this section).

The memory mapping of the 80col Video RAM and VFC ROM is controlled by I/O port $EC. It can be mapped to any 4K address boundary, and the RAM/ROM can each be enabled/disabled independently. The control is identical to the MAP80 VFC.

The external RAM can be paged in 32Kbyte or 64Kbyte chunks under the control of I/O port $FE. The control is identical to the MAP80 256K RAM board.

ROM and RAM memory resources behave somewhat differently relative to the underlying RAM:

- Assume all of the NASCOM resources are disabled (mapped out), there is no write protection and the MAP80VFC resources are mapped into the address space. The VFC ROM contains CP/M bootstrap code and low-level code for controlling the VDU. When writes are performed to the VFC (80col) video RAM, the write data only goes to the video RAM, not to the underlying external RAM.
- Assume the MAP80 VFC resources are disabled (mapped out) and there is no write protection. When writes are performed to the video RAM or workspace RAM, the write data only goes to those locations, not to the underlying external RAM. If one of other of those devices is disabled (mapped out) the write data will go to external RAM. However, when writes are performed to the NAS-SYS 3 or SBROM, the write data goes to the underlying external RAM.

As a consequence of the behaviour described above, it’s possible to copy the NAS-SYS ROM to itself (`C 0 0 800` from NAS-SYS) then disable the ROM so that NAS-SYS is running from RAM. By setting the appropriate write-protect bit this RAM image is protected from corruption.

Alternatively, it’s possible to load a new image into RAM at address 0 (for example, NAS-SYS 1) and switch to execution of this RAM image. In this case, the hand-over is not so simple; simply disabling the ROM will leave the processor PC at a location that is sensible for the old ROM but may not be sensible for the new image in RAM. The hand-over needs to involve branching out of the ROM image, disabling the ROM and then branching to an appropriate location in the new image.

There is additional hardware support to make it easier to disable the SBROM. Refer to SBROM Disable (page 27).

I/O Map

The I/O ports are summarised in the table below and described in detail in subsequent tables.

Address	Description	Where	Origin
$00	Keyboard, motor and single-step control	FPGA/External	NASCOM 1/2
$01-$02	6402 UART	FPGA	NASCOM 1/2
$04-$07	Z80-PIO control and data	External	NASCOM 1/2
$08-$0B	Z80-CTC control and data	External	(NASCOM I/O board)
$10-$17	SDcard controller $10 - $14 are used; $15 - $17 are RESERVED.	FPGA/External	NASCOM 4
$18	Memory mapping for NASCOM ROM/RAM/VDU	FPGA	NASCOM 4
$19	Memory write-protect	FPGA	NASCOM 4
$1A	Memory wait-states	FPGA	NASCOM 4
$1B	Power-on-reset high byte	FPGA	NASCOM 4
$1C	Reset reason	FPGA	NASCOM 4
$1D	Serial configuration (stop bits, polarity, baud rate)	FPGA	NASCOM 4
$E0-$E3	WD2797 Floppy Disk Controller	External	MAP80 VFC
$E4	FDC Drive select etc.	FPGA/External	MAP80 VFC
$E6	(VFC parallel keyboard) NOT IMPLEMENTED		MAP80 VFC
$E8	(VFC alarm/beeper) NOT IMPLEMENTED		MAP80 VFC
$EA	(VFC MC6845 register sel)		MAP80 VFC
$EB	(VFC MC6845 data) PART IMPLEMENTED		MAP80 VFC
$EC	VFC memory mapping	FPGA	MAP80 VFC
$EE	VFC select VFC video on primary output	FPGA	MAP80 VFC
$EF	VFC select NASCOM video on primary output	FPGA	MAP80 VFC
$FE	256K RAM paging/memory mapping	FPGA/External	MAP80 256K RAM

Port: $00 (WRITE)
Name: PORT0
Notes: The reset value is UNDEFINED but written as 0 early in the NAS-SYS (or other monitor) startup. The monitor stores a copy of the value written (“PORT0” in the workspace area) which it uses to mimic read-modify-write operations (eg, in the NAS-SYS MFLP anf FFLP SCALs).

Bit	Reset	Name	Description
7			
6			
5			Unused (on the NASCOM 1/2 this is wired to the keyboard, but not used there).
4		DRIVE	Turn on the Tape “DRIVE” LED.

3		NMI	Generate an NMI (used for single-step).
2			Unused (on the NASCOM 1/2 this is wired to the keyboard, but not used there).
1		KBDRST	Reset the row[2] counter on the NASCOM keyboard.
0		KBDCLK	Clock the row counter on the NASCOM keyboard.

Port: $00 (READ)
Name: PORT0
Notes: Read current state of keyboard

Bit	Reset	Name	Description
7:0	?	KBD	Current state of the currently-selected column of the keyboard. 1 represents a released key, 0 represents a pressed key.

Port: $01
Name: UART DATA
Notes: When data available, read returns read data. When space available, sends write byte for serial transmission.

UART serial format is fixed at 8 bits in hardware; 7-bit data with parity is supported under software control in NAS-SYS.

Stop bits can be 1 or 2 (selected by LKSW1) on NASCOM 1/2, programmable (Port $1D) on NASCOM 4.
Receive buffering is 1 byte on NASCOM 1/2, 16 bytes on NASCOM 4.

Port: $02 (READ-ONLY)
Name: UART STATUS
Notes: NAS-SYS uses the RXD and TXE bits but ignores any errors flagged in bits [3:1].

Bit	Reset	Name	Description
7		RXD	1: Receive data available
6		TXE	1: Tx buffer empty (can accept Tx data byte)
5			Unused: read 0.
4			Unused: read 0.
3		RXFE	1: Receive framing error
2		RXPE	1: Receive parity error
1		RXOV	1: Receive overrun error

2 The words “row” and “column” here correspond to the usage in the NAS-SYS source code, which is opposite to the way that the keyboard schematic is drawn. On the schematic, the counter selects one of the “Drive” lines, which are drawn vertically (columns) and the keys of the driven column induce a response in one or more of the “Sense” lines, which are drawn horizontally (rows).

0			Unused: read 0.

Port: $04 - $07
Name: Z80 PIO
Notes: Refer to Z80 PIO data sheet for programming information. Supports vectored interrupts. The Z80 PIO and Z80 CTC form an interrupt daisy-chain (using IEI/IEO) with the Z80 PIO in the first (highest-priority) position.

Port: $08 - $0B
Name: Z80 CTC
Notes: Refer to Z80 PIO data sheet for programming information. Supports vectored interrupts. The Z80 PIO and Z80 CTC form an interrupt daisy-chain (using IEI/IEO) with the Z80 PIO in the first (highest-priority) position.

Port: $10 (READ/WRITE)
Name: SDDATA
Notes: SDcard data register. Refer to SDcard Interface (page 26).

Port: $11 (READ)
Name: SDSTATUS
Notes: SDcard status register. Refer to SDcard Interface (page 26).

Bit	Reset	Name	Description
7		WRRDY	1: Write data byte can be accepted.
6		RDRDY	1: Read data byte is available.
5		BUSY	1: Block busy.
4		INIT	1: Init busy.
3:0			Unused: read 0.

Port: $11 (WRITE)
Name: SDSTATUS
Notes: SDcard status register. Refer to SDcard Interface (page 26).

Bit	Reset	Name	Description
7:1			Unused: write data ignored.
0		CMD	0: Read block 1: Write block

Port: $12 (WRITE-ONLY)
Name: SDLBA0
Notes: SDcard address register. Specifies bits [7:0] of the logical block address. Refer to SDcard Interface (page 26).

Port: $13 (WRITE-ONLY)
Name: SDLBA1
Notes: SDcard address register. Specifies bits [15:8] of the logical block address. Refer to SDcard Interface (page 26).

Port: $14 (WRITE-ONLY)
Name: SDLBA2
Notes: SDcard address register. Specifies bits [23:16] of the logical block address. Refer to SDcard Interface (page 26).

Port: $18 (READ/WRITE)
Name: REMAP
Notes: The reset value is UNDEFINED. These bits allow address space to be reconfigured either to allow CP/M operation or to allow the monitor ROM to be replaced with RAM.

Bit	Reset	Name	Description
7	0		Unused: write data ignored, read 0.
6	0	CHARGEN	0: VFC ROM/RAM regions behave normally. 1: VFC RAM region occupies the whole 4Kbyte window of the VFC and provides write-only access to the character generator (see Programmable Character Generator on page 32)
5	0	VFCAUTO	MAP VFC autoboot. Determines effect of VFCMAP[1] (Port $EC). This mimics the function of the L4 jumper on a MAP80 VFC board.
4	0	NASWSRAM	1: Enable NASCOM workspace RAM; 1Kbytes at $0C00.
3	0	NASROM	1: Enable NASCOM ROM monitor; 2Kbytes at $0000.
2	1	SBROM	1: Enable special boot ROM; 1Kbytes ar $1000. The disable for this ROM has special behavior; refer to SBROM Disable (page 27).
1	0	NASVRAMHI	0: NASCOM video RAM is decoded at $0800. 1: NASCOM video RAM is decoded at $F800 (for NASCOM CP/M).
0	0	NASVRAM	1: Enable NASCOM video RAM; 1Kbytes.

Port: $19 (READ/WRITE)
Name: PROTECT
Notes: The reset value is UNDEFINED. The write-protect address regions correspond to the NASCOM 2 monitor ROM, EPROMs on the NASCOM 2 board and the NASCOM 2 BASIC ROM.

Bit	Reset	Name	Description
7	0		Unused: write data ignored, read 0.

6	X	PROTEF8K	1: Write-protect 8Kbyte region starting from $E000
5	X	PROTDF4K	1: Write-protect 4Kbyte region starting from $D000
4	X	PROTCF4K	1: Write-protect 4Kbyte region starting from $C000
3	X	PROTBF4K	1: Write-protect 4Kbyte region starting from $B000
2	X	PROTAF4K	1: Write-protect 4Kbyte region starting from $A000
1	0		Unused: write data ignored, read 0.
0	X	PROT0F2K	1: Write-protect 2Kbyte region starting from $0000

Port: $1A (WRITE-ONLY)
Name: MWAITS
Notes: Controls the number of wait states inserted on memory read and write operations. Reset to $20, which gives a performance approximately equal to running a Z80 at 4MHz. All values (0-255) are legal.

Port: $1B (READ/WRITE)
Name: PORPAGE
Notes: This register has no hardware side-effect. The reset value is UNDEFINED. It is managed by SBROM. Refer to Reset (cold and warm) and boot (page 27)

Port: $1C (READ, WRITE-1-TO-CLEAR)
Name: REASON
Notes: Bits 7, 6 behave identically in hardware and are managed by SBROM to implement the described functionality TODO additional functionality (reason codes) is planned but not yet implemented.

Bit	**Reset**	**Name**	**Description**
7		COLD	0: Warm reset 1: Cold reset
6	0	NEVERBOOTED	0: Booted successfully 1: Never booted
5	0		Unused: read 0.
4	0		Unused: read 0.
3	0		Unused: read 0.
2	0		Unused: read 0.
1	0		Unused: read 0.
0	0		Unused: read 0.

Port: $1D (WRITE-ONLY)
Name: SERCON
Set data rate for Rx and Tx on SERIAL connector (NASCOM 2 allows 300baud, 1200baud or external clocking to be selected by LSW1; with some additional wires, it can also generate

2400baud).
Set data polarity
Set number of stop bits (NASCOM 2 allows 1 or 2 selected by LSW1/5).
Notes: The reset value is UNDEFINED. The reset values shown below are established by SBROM as part of a cold reset.

Bit	Reset	Name	Description
7:4	0	STOP	0: 1 stop bit 1: 1.5 stop bits 2: 2 stop bits 3: RESERVED.
5	0		Unused: write data ignored.
4	0	IDLELOW	0: Serial data lines idle at logic ‘1’ 1: Serial data lines idle at logic ‘0’.
3:0	7	SERSPEED	0: 300baud [1] 1: 1200baud 2: 2400baud 3: 4800baud 4: 9600baud 5: 19200baud 6: 38400baud 7: 115200baud (reset default) 8: Use external 16x baud rate clocks. TODO [1] not currently working,

Port: $E0 - $E3
Name: WD2797 Floppy Disk Controller
Notes: Refer to Western Digital WD2797 data sheet for programming information.

$E0: FDC command (write) FDC status (read)
$E1: FDC track register
$E2: FDC sector register
$E3: FDC data register

Port: $E4 (WRITE)
Name: DRIVE SELECT
Notes: Each write to this register (re)triggers the monostable that keeps the drive motor running. The first write (the write that starts the motor) triggers the local drive-ready monostable. Select for Drive 3 only operates if the jumpers are set appropriately on the NASCOM 4.

Bit	Reset	Name	Description
7	?		Unused: write data ignored.
6	?		Unused: write data ignored.
5	?	CLKDOUBLE	Double clock speed, for 8” drive support
4		FM/MFM	Density. 0: MFM (double), 1: FM (single)

3		SEL3	Select drive 3
2		SEL2	Select drive 2
1		SEL1	Select drive 1
0		SEL0	Select drive 0

Port: $E4 (READ)
Name: DRIVE STATUS
Notes: The /READY signal can be generated by the NASCOM 4 or by the drive, depending upon the jumper settings on the NASCOM 4.

Bit	Reset	Name	Description
7	?	DRQ	Data request from FDC
6	0		Unused: read 0.
5	0		Unused: read 0.
4	0		Unused: read 0.
3	0		Unused: read 0.
2	0		Unused: read 0.
1	?	/READY	Drive not ready from drive/controller support logic
0	?	INTRQ	Interrupt from FDC

Port: $EA (WRITE-ONLY)
Name: CRTCREG
Notes: This implements the MC6845 address register. The low 5 bits are read/write and affect what happens when you access $EB.

Port: $EB (WRITE-ONLY)
Name: CRTCDAT
Notes: Partial implementation of the MC6845 data register. Only the following registers are implemented:
R10 Cursor start. Bits 6:5 control cusor visibility/blink, bits 4:0 select cursor start scan row. WO.
R11 Cursor end. Bits 4:0 select cursor end scan row. WO.
R14 Cursor address high. WO (R/W on real MC6845).
R15 Cursor address low. WO (R/W on real MC6845).
The (hardware) cursor only exists on the MAP80 VFC 80-column display
TODO The MAP80VFC ROM writes to 14 then 15, and never reads. It may be necessary to hardware buffer the high write to stop the cursor from briefly bouncing to an intermediate location. Check the MC6845 data sheet on this topic.
TODO implementation in progress but not yet complete.

Port: $EC (WRITE-ONLY)
Name: VFCMAP
Notes: Controls location of VFC in the address space. On a MAP80 VFC board, L4 controls

whether the board autoboots by enabling its ROM at address 0 after reset. On NASCOM 4, REMAP[5] is controlled by the SBROM to perform the same function.

Bit	Reset	Name	Description
7:4	0	VFCPAGE	Select 4Kbyte boundary for VFC
3		EPROM2	Not implemented on NASCOM 4. On the real VFC board this allowed an alternative character generator ROM to be selected.
2	0	INVVIDEO	0: Characters 128-255 are block graphics characters etc. 1: Characters 128-255 are inverse-video versions of characters 0-127.
1	0	VSOFT	Enable VFC “VSOFT” ROM in low half of 4Kbyte block. The actual ROM enable is the XOR of this bit with REMAP[5]. Therefore: REMAP[5]=0: the VSOFT ROM is disabled after reset, writing VSOFT=1 enables the ROM. REMAP[5]=1: the VSOFT ROM is enabled after reset, writing VSOFT=1 *disables* the ROM.
0	0	VIDRAM	Enable VFC video RAM in high half of 4Kbyte block

Port: $EE
Name: VIDVFC
Notes: A read from or write to this port (any data) selects the MAP80 VFC 80-column video as the output on the primary VGA connector and NASCOM 48x16 video on the secondary VGA connector.

On a MAP80 VFC board, L2 controls which port selects which video output, and so the assignment here between $EE and $EF is somewhat arbitrary. The programming example in Section 4 of the VFC documentation suggests that port $EE should select the NASCOM video output. However, the MAP80 CP/M 2.2 BIOS code does a read from $EE as part of its VFC init, so using that port to select VFC output provides the best software compatibility. (The MAP80 CP/M 3 code does *not* access either $EE or $EF).

Port: $EF (WRITE)
Name: VIDNAS
Notes: A read from or write to this port (any data) selects the NASCOM 48x16 video as the output on the primary VGA connector and MAP80 VFC 80-column video on the secondary VGA connector.

Port: $FE (WRITE-ONLY)
Name: MAPRAM
Notes: In order to use paging with 64Kbyte pages is it necessary to have some kind of software “kernel” in overlying memory (eg one of the pageable NASCOM memories).

1 external RAM provides 512Kbytes of memory (8, 64Kbyte pages or 16, 32Kbyte pages).

2 external RAMs provide 1024Kbytes of memory (16, 64Kbyte pages or 32, 32Kbyte pages).			
Bit	Reset	Name	Description
7	0	PAGESIZ32K	0: Use 64Kbyte pages 1: Use 32Kbyte pages
6	0	UPPER32K	When PAGESIZ32K=1 0: The lower 32Kbytes of the address space decodes page 0 and is fixed, the upper 32Kbytes is paged. 1: The upper 32Kbytes of the address space decodes page 1 and is fixed, the lower 32Kbytes is paged.
5:0	0	PAGESEL	Select memory page. When PAGESIZ32=0 (64Kbyte pages) the pages are 0, 2, 4, 6 etc (the LSB is implicitly 0 and is ignored). When PAGESIZ32=1 (32Kbyte pages) the pages are 0, 1, 2, 3 etc.

SDcard Interface

The SDcard interface is handled in hardware within the FPGA. The interface supports standard capacity (SDSC) and high capacity (SDHC) cards.

The interface supports read and write of a 512-byte block at a specified Logical Block Address (LBA).

After reset, wait until SDSTATUS returns a value of $80.

To read a 512-byte block from the SDcard:

- Wait until SDSTATUS=$80 (ensures previous command has completed)
- Write the block address to LBA2, LBA1, LBA0 (you only need to write values that have changed since the last command. For example, if LBA2 remains at 0, there is no need to write it again)
- Write 0 to SDCTRL to issue the READ command
- Option 1: use counter to loop 512 times: wait until SDSTATUS=$E0 (read byte ready, block busy), read byte from SDDATA. Loop.
- Option 2: if SDSTATUS=$80 done (512 bytes read) else if SDSTATUS=$E0 (read byte ready, block busy), read byte from SDDATA. Loop.

To write a 512-byte block to the SDcard:

- Wait until SDSTATUS=$80 (ensures previous command has completed)
- Write the block address to LBA2, LBA1, LBA0 (you only need to write values that have changed since the last command. For example, if LBA2 remains at 0, there is no need to write it again)
- Write 1 to SDCTRL to issue the WRITE command

- Option 1: use counter to loop 512 times: wait until SDSTATUS=$A0 (ready for write byte, block busy), write byte to SDDATA.
- Option 2: if SDSTATUS=$80 done (512 bytes written) else if SDSTATUS=$A0 (write data byte can be accepted, block busy), write byte to SDDATA. Loop.

For both reads and write, it is always necessary to transfer 512 bytes. If necessary, read bytes and discard them, or write dummy bytes.

For code examples, refer to the subroutine SDRD512 in the SBROM source code, or the code in the PolyDos ROM (polydos_rom.asm) or the code in the CP/M BIOS (n4bios.mac)

SBROM Jump-on-reset

The SBROM is enabled automatically at reset, and is decoded at address $1000. Jump-on-reset logic in the FPGA selects the SBROM for the first few M1 cycles after reset (similarly to the circuitry on the NASCOM 2). For correct behaviour, it is necessary for the SBROM code at $1000 to be a JP instruction (JR is no good, because it would calculate its destination relative to the CPU's idea of the PC).

SBROM Disable

A typical function for this ROM is to load images to memory (for example, to load ROM BASIC to the top of the address space) and then to disable itself and jump into NAS-SYS 3. In an early version of the design this was achieved by building a blob of code on the stack and branching to it from the SBROM. The code executing in stack memory would disable the SBROM and jump to NAS-SYS 3. That worked successfully but had the effect of modifying some memory (those few stack locations).

To achieve the warm-start functionality, the current version allows SBROM to disable itself without making any memory modifications.

When the SBROM bit of the REMAP port is written with 0, the ROM disable is delayed. The delay is designed so that, if the following code executes from the SBROM and the OUT instruction sets the SBROM bit to 0, the ROM will remain enabled until the the JMP and its operands have been read from the SBROM:

```
OUT (REMAP), A
JP  (HL)
```

Reset (cold and warm) and boot

There are four sources of reset:

- Power-on reset (performs a cold reset)
- Reset from the RESET or WARM RESET buttons on the front edge of the PCB
- Reset from the cold (F1) or warm (F2) function keys on the PS/2 keyboard TODO not yet implemented
- Reset from the (cold) RESET button on the NASCOM2 keyboard

All of these have the same effect: they all perform a hardware reset. The difference between a cold and warm reset is that cold resets set bits 7, 6 in the REASON register (port $1C). This allows software (the SBROM) to distinguish them and behave differently.

At reset, the Special Boot ROM (SBROM) is enabled in the memory map: 1Kbytes at address $1000. A jump-on-reset circuit (just like on a NASCOM 2) makes the Z80 fetch its first instruction from $1000 and so execution passes to the SBROM.

The reason for decoding the SBROM at $1000 is simple: it allows the SBR to initialise NAS-SYS 3 and to make use of its I/O routines. This makes the SBR code much smaller and simpler.

After a cold reset, the SBROM displays a boot menu (loaded from SDcard) which allows the user select how the system will be configured. Once a menu selection is made, the usual result is that zero or more images are loaded from SDcard to memory, the PROTECT and REMAP ports are written and control is passed (using a Z80 JP) to a new location. Before the final JP, the NeverBooted bit is cleared and the high byte of the destination is written to the PORPAGE port.

After a warm reset, the SBROM checks the NeverBooted bit. If that bit is clear, it implies that the PORPAGE port value is valid. In this case, the SBROM does as little as possible; it assumes the memory is already set up. The REMAP, PROTECT and PORPAGE ports are not affected by reset. The SBROM reads the value of PORPAGE, pads it with a byte of $00 and jumps there, disabling itself on the way (the REMAP value is generated by reading the current value and masking out the SBROM enable bit). This process performs no memory writes and makes no use of the stack. It aims to be fast, invisible and non-intrusive.

If the NeverBooted bit is set after a warm reset, it implies that the SDcard is absent or cannot be read. In this case, the SBROM jumps to the NAS-SYS3 image that is part of the FPGA. If external memory is fitted, it will be available.

The Special Boot ROM (SBROM)

The SBROM is responsible for cold and warm reset. In general, the final function of the SBROM is to disable itself and pass control to some other piece of code. However, it is possible to map it back into the memory map, as long as it does not collide with another piece of code (SBROM is decoded in the region $1000-$13FF). It contains some utility routines that could be useful, as well a subroutine for reading the SDcard. Refer to the source code for details.

The Menu System

The menu system is implemented by the SBROM and provides a *very* crude scripting facility. The menu text itself is loaded from SDcard and written to the NASCOM screen (the 48x16 screen).

The menu system supports 26 entries, selected using the letters A-Z (upper case only). Once a letter is pressed, its menu actions are performed immediately; there is no need to press return.

Each of the 26 menu entries has an associated 512-byte block on the SDcard, called a “profile”. The profile is loaded into memory and interpreted by code in the SBROM. In order to minimise the disruption to system memory, the profile is loaded into the display memory, overwriting the top of

the screen and the menu itself; you can see it there for a fraction of a second while the boot is happening.

The profile can contain any sequence of the following commands, in ASCII text:

- Wxxxx=yyyy – write to address xxxx with data yyyy. Address and data are in hex. Each can be between 1 and 4 characters in length (no padding is required).
- Pxx=yy – write to I/O port xx with data yy. Port and data are in hex. Each can be between 1 and 2 characters in length (no padding is required).
- Ixxxx – prepare to load image starting from block xxxx. Block is in hex. Can be between 1 and 4 characters in length (no padding is required).
- Lxxxx=yyyy – load yyyy blocks from SDcard (starting at the block specified by the most recent I command) into memory starting at address xxxx. Block count and address are in hex. Each can be between 1 and 4 characters in length (no padding is required).
- Gxxxx=yy – Go. Write yy to REMAP and then jump to address xxxx. Data and address are in hex. Each can be between 1 and 4 characters in length (no padding is required).

Each command (including the last) is terminated by 1 space. The last command is required to be a G and therefore anything after the G's terminating space is ignored. However, for politeness, the profile should be padded with spaces.

Menu Examples

The make_rom_menu script (written in PERL) creates a lump of binary data containing the menu, profiles and ROM images. This data is combined with virtual disk images to create the complete SDcard image (see Creating and manipulating SDcard images on page 37). make_rom_menu is not interactive; it is configured by editing its three main sections:

- Text definition of the menu
- List of images and their locations on the host system
- List of profile definitions

The menu definition is simple; a single string:

```
#    12345678901234567890123456789012345678901234567890123456789012345678
my $menu =
    "A: T2  B: BBUG  C: T4  D: NAS-SYS1  E: NAS-SYS3\r" .
    "F: T4       + ZEAP + BASIC\r" .
    "G: NAS-SYS1 + ZEAP + BASIC\r" .
    "H: NAS-SYS3 + ZEAP + BASIC + DISDEBUG\r" .
    "I: NAS-SYS3 + BASIC + POLYDOS2  J: POLYDOS2-SD\r" .
    "K: NAS-SYS3 + BASIC + POLYDOS3  L: POLYDOS-LSD\r" .
    "M  NAS-SYS3 + BASIC + NASDOS    N: NASDOS-SD\r" .
    "O: NAS-SYS3 + PASCAL\r" .
    "P: NASCOM CP/M\r" .
    "Q: MAP80 VFC CP/M      R: MAP80 VFC CP/M-LSD\r" .
    "S: NAS-SYS3 + LOLLIPOP\r" .
    "T: NAS-SYS3 + INVADERS\r" .
    "U: NAS-SYS3 + NASPEN\r" .
    "V: NAS-SYS3 + MEMORY TEST\r" ;
```

The list of images looks like this:

```
add_image("NASBUGT2", 0x0000, "$nascom/ROM/nasbugt2/NASBUGT2.bin_golden");
add_image("NASBUGT4", 0x0000, "$nascom/ROM/nasbugt4/NASBUGT4.bin_golden");
add_image("BBUG",     0x0000, "$nascom/ROM/bbug/BBUG.bin_golden");
add_image("NASSYS1",  0x0000, "$nascom/ROM/nassys1/NASSYS1.bin_golden");
add_image("NASSYS3",  0x0000, "$nascom/ROM/nassys3/NASSYS3.bin_golden");
add_image("BASIC",    0xE000, "$nascom/ROM/basic/basic.nas");
add_image("ZEAP",     0xD000, "$nascom/ROM/zeap/zeap.nas");
etc..
```

Each image is given a name for use within the script and a load address. Binary and .NAS files are supported (.NAS files are automatically converted to binary using the nascon utility). Each image is loaded and padded to a multiple of 512bytes (if necessary). The images are then sized and assigned a start block number on the SDcard.

The list of profile definitions (currently) looks like this:

```
add_profile('A', "L:NASBUGT2",            "P19=01", "G0=11");
add_profile('B', "L:BBUG",                "P19=01", "G0=11");
add_profile('C', "L:NASBUGT4",            "P19=01", "G0=11");
add_profile('D', "L:NASSYS1",             "P19=01", "G0=11"); # NAS-SYS1 in RAM
add_profile('E', "L:NASSYS3",             "P19=01", "G0=11"); # NAS-SYS3 in RAM
add_profile('F', "L:NASBUGT4", "L:ZEAP", "L:BASIC",        "P19=61", "G0=11");
add_profile('G', "L:NASSYS1", "L:ZEAP",  "L:BASIC",        "P19=61", "G0=11");
add_profile('H', "L:DISDEB", "L:ZEAP",  "L:BASIC", "P19=70", "G0=19");
add_profile('I', "L:POLYDOS2", "L:BASIC",          "P19=60", "GD000=19");
add_profile('J', "L:POLYDOS2-SD", "L:BASIC",       "P19=60", "GD000=19");
add_profile('K', "L:POLYDOS3", "L:BASIC",          "P19=60", "GD000=19");
add_profile('L', "L:POLYDOS-LSD", "L:BASIC",       "P19=60", "GD000=19");
add_profile('M', "L:NASDOS",   "L:BASIC",          "P19=60", "GD000=19");
add_profile('N', "L:NASDOS-SD","L:BASIC",          "P19=60", "GD000=19");
add_profile('O', "L:PASCAL",         "P19=60", "GD000=19");
add_profile('P', "L:NASCPM",        "GF000=3");
add_profile('Q',       "PEE=00", "G0=20");   # MAP80 CP/M  - switch video
add_profile('R', "L:CPM-LSD",  "PEE=00", "GE000=20");# MAP80 CP/M  - switch
video, SDcard boot
add_profile('S', "L:LOLLIPOP",                                "G1000=19");
add_profile('T', "L:INVADERS",                                "G1000=19");
add_profile('U', "L:NASPEN",                          "P19=08", "G0=19");
add_profile('V', "L:MEMTEST",              "G0=19"); # Load then start NAS-SYS
```

Each definition starts with a letter (the letter that will select it in the menu) followed by a comma-separated list of commands. The P, W and G commands are encoded in the profile exactly as they are written above (there is no example of P). The “L:<name>” is translated into an I and an L command in the profile: within make_rom_menu, the symbolic name (eg ZEAP) created by add_image() provides the image size, location on SDcard and load address.

When executed, make_rom_menu creates a binary file and displays progress messages that include the converted/encoded profiles (the profile padding is not shown):

```
INFO profile A as I22 L0=2 P19=01 G0=11
INFO profile B as I28 L0=4 P19=01 G0=11
INFO profile C as I24 L0=4 P19=01 G0=11
INFO profile D as I2C L0=4 P19=01 G0=11
INFO profile E as I30 L0=4 P19=01 G0=11
```

```
INFO profile F as I24 L0=4 I44 LD000=8 I34 LE000=10 P19=61 G0=11
INFO profile G as I2C L0=4 I44 LD000=8 I34 LE000=10 P19=61 G0=11
INFO profile H as I8E LC000=8 I44 LD000=8 I34 LE000=10 P19=70 G0=19
INFO profile I as I50 LD000=4 I34 LE000=10 P19=60 GD000=19
INFO profile J as I54 LD000=4 I34 LE000=10 P19=60 GD000=19
INFO profile K as I5C LD000=4 I34 LE000=10 P19=60 GD000=19
INFO profile L as I58 LD000=4 I34 LE000=10 P19=60 GD000=19
INFO profile M as I78 LD000=8 I34 LE000=10 P19=60 GD000=19
INFO profile N as I80 LD000=8 I34 LE000=10 P19=60 GD000=19
INFO profile O as I60 LD000=18 P19=60 GD000=19
INFO profile P as I88 LF000=4 GF000=3
INFO profile Q as PEE=00 G0=20
INFO profile R as I9F LE000=1 PEE=00 GE000=20
INFO profile S as I8C L1000=2 G1000=19
INFO profile T as I96 L1000=7 G1000=19
INFO profile U as I4C LB800=4 P19=08 G0=19
INFO profile V as I9D LC80=2 G0=19
```

Looking at one profile in detail (profile H):

```
I86 LC000=8 I44 LD000=8 I34 LE000=10 P19=70 G0=19
```

Each of the “Ixx Lxxxx=yyyy” command-pairs is loading an image into memory. The first loads 8 blocks (4Kbytes) to memory at $C000, the second loads 8 blocks to memory at $D000 and the third loads 16 blocks (8Kbytes) to memory at $E000. Referring back to the associated “add_profile()” definition above suggests that these images are DISDEBUG, ZEAP and BASIC, respectively. The command “P19=70” performs a write to I/O port PROTECT: it is write-protecting the 3 memory regions that have just been loaded, making them look like ROM. Finally, the “G0=19” command does a jump to address 0 (to NAS-SYS 3) and writing $19 to I/O port REMAP. Referring to I/O Map (page 18) shows that this is enabling NASCOM ROM, Video RAM and workspace RAM, and disabling SBROM.

Limitations

The menu loading system has these known limitations:

- A single screen isn’t really big enough, so the menu is a bit cramped
- Each loaded image is a multiple of 512 bytes long. For the case where an image is to be loaded into NASCOM workspace RAM at $0C80, it’s difficult to load an image without overwriting the stack (which crashes SBROM)
- Use of a raw format makes it trickier for some users to load images onto an SDcard. A basic RO FAT implementation might be worthwhile.

The Video Sub-system

The board provides 2 character-mapped video sub-systems which operate simultaneously.

The NASCOM video provides a 64 character x 16 line output, but 16 characters of each line are hidden, so that the effective output is 48 x 16. The video RAM is 1Kbytes. It is mapped to a start address of $0800 by default but it can be disabled from the memory map or remapped to a start

address of $F800[3] by writing to port $18. The layout of the video memory is documented at the back of the NAS-SYS 3 documentation.

The VFC video provides an 80 character x 25 line output. The video RAM is 2Kbytes. It is addressed in the top half of the MAP80 VCF 4Kbyte address space, which is itself pageable by wrtiting to port $EC. The video uses 80x25=4000 locations of the 4096-byte video RAM. The layout is straightforward: the first 80 bytes correspond to the top line, etc.

Both video outputs generate a monochrome output to drive a VGA multi-sync monitor. There are 2 VGA connectors on the board and the outputs can be swapped between the connectors by writing to port $EE. Resistors on the PCB allow the monochrome display to drive any single colour by mixing proportions of the RGB signal. For example: white, green or amber can be generated.

Both video sub-systems use dual-ported memories within the FPGA and so there is no contention between CPU and video access (no NASCOM 1-style white snow and no NASCOM 2-style black over-blanking).

Both video sub-systems use the same character size: 8 pixels horizontally by 16 rows vertically. Characters abut horizontally and vertically; any inter-character spacing must be incorporated within the character-set design. Each character requires 16 bytes of storage and there are 256 possible characters so the character generator ROM is 16*256=4Kbytes in size. The character generator is shared between the two video sub-systems, so any given character will appear identical on either output. Sharing is arbitrated so that there is no contention between NASCOM video and VFC video access to the character generator ROM.

The NASCOM 1 (and NASCOM 4) displays all 16 rows of the character on the video output. The NASCOM 2 only displays the top 12 or 14 rows of the character were displayed (controlled by LSW 1/6). The motivation was to fit the whole of the display onto the screen of a domestic TV,

The NASCOM video has a native size of 48x8=384pixels x 16x16=256lines. It uses a VGA mode with a resolution of 800x600@50MHz. Within the FPGA, each native pixel is duplicated and each line is duplicated, so that the NASCOM video occupies 768pixels x 512lines of the 800x600 screen.

The VFC video has a native resolution of 80x8=640pixels x 25x16=400 lines. It uses a VGA mode with a resolution of 640x400@25MHz. Within the FPGA, the video sub-system runs at 50MHz and so each native pixel is duplicated so that it occupies the correct amount of time horizontally.

Programmable Character Generator

In the previous section the character generator was referred to as a 4Kbyte ROM. However, on an FPGA the memory structures are all inherently RAM devices: a ROM is simply a RAM with no write port and with contents initialised from the FPGA programming bitstream. As a result, it is trivial to make the character generator re-programmable under software control. Such programmable character generators existed as add-ons to the NASCOM but the behaviour of this one is not based on any earlier design.

3 NASCOM supplied a version of CP/M that used the NASCOM 2 video. It required a modified boot ROM and NAS/MD PROM. The PROM decoded the boot ROM at $FC00 and the video RAM at $F800.

The character generator can decoded in the address space associated with the VFC. Therefore, to access the character generator, a 2-step process is required. In either order:

- Map the 4Kbyte VFC into the address space using a write to port $EC, enable VIDRAM.
- Set the CHARGEN bit using a write to port $18

CPU access to the character generator is write-only; reads will return invalid data

The initial content of the character generator (established by the FPGA configuration bitstream) corresponds to the NASCOM NAS-CHAR and NAS-GRA character sets. A reset of the board does not cause the FPGA bitstream to be reloaded. Therefore, any programmed changes to the character generator will remain after reset. To restore the default character generator contents, either power-cycle the board or select a menu item from the boot menu that reloads the character generator from an image on the SDcard.

The CPU and video sub-systems share access to the address port of the character generator ROM. There is no arbitration and CPU access takes priority. As a result, there may be transient corruption of the video image (“snow”) when the CPU writes to the character generator.

Programmable Character Generator Example

This example uses NAS-SYS 3 commands to modify characters in the character generator. Each character occupies 16 consecutive bytes. A byte of 0 in the display memory corresponds to the first 16 bytes in the character generator; a byte of 1 to the second 16 bytes, etc.

Design a character by drawing out an 8x16 grid then encode as a series of bytes in which a binary “1” appears where a lit pixel is required. Here’s an example, chosen for its lack of symmetry:

```
00000111    $07
00011000    $18
00100000    $20
01000011    $43
10000000    $80
10000000    $80
10000000    $80
11110000    $F0
00001000    $08
10000100    $84
01000010    $42
01000010    $42
00100100    $24
00011000    $18
00011000    $18
00011000    $18
```

Here’s the command sequence to program it into memory from NAS-SYS 3:

```
O EC E1                          map VFC into memory at $E000
M E800
 00 01 02 03 04 05 06.           modify top line of VFC display
                                 (look at the VFC output to view this edit)
M 0BCA
 00 01 02 03 04 05 06.           modify top line of NASCOM display
                                 (it will be visible on the screen)
O 18 59                          enable character generator access
```

```
M E000
 AA AA AA AA AA AA AA AA            character 0: vertical stripes
 AA AA AA AA AA AA AA AA
 07 18 20 43 80 80 80 F0            character 1: the pattern from above
 08 84 42 42 24 18 18 18.
O 18 19                             disable character generator access
O EC E0                             unmap VFC from memory
```

Disk Operating Systems

This section provides some notes on running the various disk operating systems that existed on the NASCOM. Please contribute additional information/configurations or ask me if you get stuck.

PolyDos

There are 3 ways to run PolyDos on the NASCOM 4:

- Using the FDC and original magnetic media or Gotek for PolyDos2 or PolyDos 3. The boot ROMs are part of the menu system: options POLYDOS2, POLYDOS3.
- Using my nascom_sdcard board attached to the PIO. The boot ROM is part of the menu system: option POLYDOS2-SD. Refer to the nascom_sdcard user guide for more details.
- Using the local SDcard to store disk images. This allows PolyDos to run on a “Stage 2” build, without the need for PIO or FDC. The boot ROM is part of the menu system: option POLYDOS-LSD. This requires the SDcard to contain appropriate disk images. See Creating and manipulating SDcard images on page 37.

PolyDos Utilites

There are 2 utilities for the Local SDcard (LSD) port:

- SETDRV – display and change virtual disks (nascom_sdcard has a utility of the same name but the code is different)
- CASDSK – Intercept NAS-SYS Read/Write commands (this is identical to the same-named nascom_sdcard utility)

SETDRV

Display and change virtual disks. There are 4 drives (0-3) and 16 virtual disks (0-9, a-f). At boot, disks 0-3 are mounted in drives 0-3.

```
$ SETDRV
```

Reports the disk mounted in each drive

```
$ SETDRV n m
```

Unmount any disk currently associated with drive n and mount disk m instead. For example:

```
$ SETDRV 1 6
```

In this example, drive 1 is now associated with virtual drive 6.

It is legal but unwise to mount the same disk in multiple drives. The drive selectd as the boot drive must always be associated with a disk that contains PolyDos system files.

CASDSK

Allows disk load/store for a program that was designed to use the `W` and `R` tape routines.

Intercepts the NAS-SYS `W` and `R` routines and redirects them to a single pre-defined disk file. Acts as a "terminate and stay resident" program and therefore must sit in free memory somewhere.

Example: Colossal cave adventure can "save" the game state using tape routines. Do this:

```
$ CASDSK CAVE.ME
Installed
$ COLOSSAL
```

Using SAVE and RESTORE within the program will still call `W` and `R` but now:

- `W` (write to tape) will actually write to the file `CAVE.ME`, deleting any pre-existing file of that name.
- `R` (read from tape) will result in the contents of `CAVE.ME` being loaded into memory at the address from which it was saved.

When installed and executed with no arguments, `CASDSK` is uninstalled; the normal `R` and `W` vectors are restored; the memory used by `CASDSK` can now be reused:

```
$ CASDSK
Uninstalled
```

When not currently installed and executed with no arguments, `CASDSK` just displays a message and terminates:

```
$ CASDSK
Not installed
```

`CASDSK` Implementation notes:

- The usual operation of PolyDos is to read and write data with a minimal granularity of 256 bytes. When saving, the same approach is taken: the write data is rounded up to the nearest 256 bytes. However, that may not be acceptable on reads, because it may overwrite data in memory. Therefore, on writes, the valid data size in the final sector (1-256 bytes) is stored in the low byte of the "execution address" entry of the data file (`CAVE.ME` in the example above). On reads, this size byte is used to transfer the file size.
- The algorithm can support any file size but the NAS-SYS calls that are intercepted are limited to a maximum size of 64Kbyte.
- This utility would work just as well on a real floppy-disk version of PolyDos.

CP/M

There are 3 ways to run CP/M on the NASCOM 4:

- Using the FDC and original magnetic media or Gotek with MAP80 VFC port. The boot ROM is part of the menu system: option MAP80 VFC CP/M.
- Using the FDC and original magnetic media or Gotek with NASCOM port. This uses the 48-column display. The boot ROM is part of the menu system: option NASCOM CP/M. This option is currently untested. Contact me if you have success or problems.
- Using the local SDcard to store disk images. This allows CP/M to run on a “Stage 2” build, without the need for PIO or FDC. The boot ROM is part of the menu system: option CP/M-LSD. This requires the SDcard to contain appropriate disk images. See Creating and manipulating SDcard images on page 37.

CP/M Utilites

There is 1 utility for the Local SDcard (LSD) port:

- SETDRV – display and change virtual disks (PolyDos has a utility of the same name but the code is different)

SETDRV

Display and change virtual disks. There are 2 drives (A, B) and 16 virtual disks (0-9, a-f). At boot, disks 0, 1 are mounted in drives A, B.

```
$ SETDRV
```

Reports the disk mounted in each drive

```
$ SETDRV n m
```

Unmount any disk currently associated with drive n and mount disk m instead. For example:

```
$ SETDRV A 6
```

In this example, drive A is now associated with virtual drive 6.

It is legal but unwise to mount the same disk in multiple drives. The drive selected as the boot drive must always be associated with a disk that contains CP/M system files.

NASDOS

There are currently 3 ways to run NASDOS on the NASCOM 4:

- Using the FDC and original magnetic media or Gotek images. The NASDOS ROMs are part of the menu system: option NASDOS. This option is currently untested. Contact me if you have success or problems.
- Using my nascom_sdcard board attached to the PIO. My port of the NASDOS ROM is part of the menu system: option NASDOS-SD. Refer to the nascom_sdcard user guide for more details. This option has received minimal testing; contact me if you have success or problems.

- A NASDOS port that uses the local SDcard for storage would be straightforward. Contact me if you are interested in such a port.

Creating and manipulating SDcard images

The SDcard has no file-system structure imposed on it; it is accessed as a sequence of 512-byte blocks. The layout of the image is shown in the table below.

Block(s)	Description
0-7	Menu. This is just ASCII text, formatted for a 48-character screen. It contains UNIX line endings and is 0-terminated. The space reserved is bigger than a NASCOM screen, allowing it to be used for something additional later (or a menu on an 80-column screen).
8-33	26 “profiles”. Each profile can load one or more ROM images to memory, perform memory and I/O writes, and should terminate by passing control to the monitor or a loaded image.
34	First block of appended ROM images
1024 (0x400)	First block of a set of 16 PolyDOS virtual disk images. Each image occupies 1MByte (so this section occupies 16MBytes) but only 512KBytes is used in each image, so each virtual disk is 512Kbytes in size.
33,792 (0x8400)	First block of a set of 16 CP/M disk images. Each image occupies 1MByte and each virtual disk is 512Kbytes in size

Various scripts are provided to facilitate the creation and maintenance of this image.

The boot menu, profiles and ROM images are created as a single unit by the script tools/make_rom_menu (see The Menu System on page 28). It is usually smaller than 512Kbytes but is padded in the image when the other sections are present.

Empty sets of PolyDos and CP/M disk images are created by the scripts tools/make_polydos_floppy_set and tools/make_cpm_floppy_set, respectively. There is usually no reason to run either of these scripts.

The usual approach is to start with the pre-built disk image tools/nascom4_sdcard.img.

In order to transfer the image to an SDcard some form of “raw” writing utility is required. From Linux, the usual approach is to use dd (carefully!).

This image can be manipulated using tools/sdcard_editor. This can be used interactively (in which case, the ‘help’ command explains the usage and all of the commands) or by putting a sequence of commands on the command-line. sdcard_editor supports these functions:

- Replace the boot menu/profiles/ROM section in the image with a new one created by tools/make_rom_menu.
- Rxtract a single PolyDos or CP/M disk image.
- Extract all 16 PolyDos or all 16 CP/M disk images.

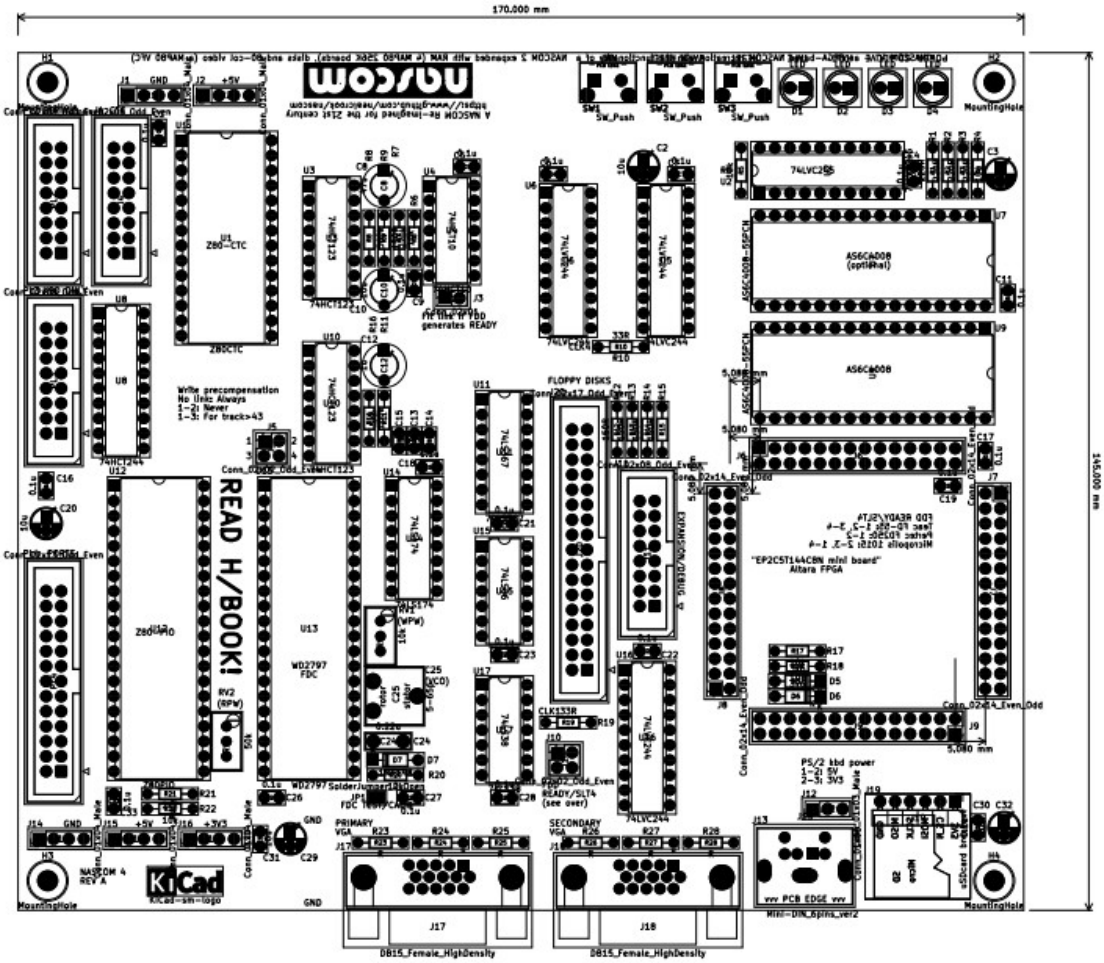
- Import a single PolyDos or CP/M disk image, overwriting the content of the current disk.
- Import all 16 PolyDos or all 16 CP/M disk images, overwriting the content of the current disks.

From Linux, sdcard_editor can be used to access/manipulate the SDcard image directly, for example:

```
$ ./sdcard_editor /dev/sdb
```

Examples of using sdcard_editor can be found in tools/README.md.

Once individual disk images have been extracted, PolyDos disk images can be manipulated using polydos_vfs and CP/M images can be manipulated using cpmtools (setups for the disk format can be found in cpm/README.md).



Construction

The PCB is double-sided with through-hole plating. The boards are 100% tested at manufacture. Unless otherwise stated, components are mounted on the top-side of the board, which is the side containing most of the silk-screen markings.

During layout, the PCB was oriented with the VGA connectors at the bottom edge and the LEDs at the top edge. (Almost) all of the component reference designators and other silk-screen markings follow this orientation and so this is how I expect that you will tend to orient the board during construction. Therefore, references to top/bottom/left/right are with respect to the top-side of the board in this orientation.

When you use the board, I expect that you will have the LEDs and buttons facing you, and so I refer to the *top* edge of the board as the *front* edge.

Components with a particular reference prefix (eg R) are numbered sequentially on the board left-to-right, top-to-bottom so that you should find the first resistor near the top-left of the board and the last near the bottom-right of the board. The exception are three resistors and one connector added for the REV B PCB, which are numbered R100, 101, 102 and J100 respectively.

I recommend that you construct your board in stages, testing it after each stage. Even if you fit all the passives and sockets in one go, I still recommend testing/adding the active components stage-by-stage. The stages are:

1. Minimal system
2. SDcard and external memory
3. Serial Port
4. I/O bus (the stages after this all require the I/O bus)
5. NASCOM Keyboard
6. PIO and/or CTC
7. Floppy Disk controller

The parts list below shows what components are required for each stage. A part marked "(*)" is optional. P: specifies lead pitch, D: specifies component diameter. For some components a part number is given: Red text is a part number from www.cpc.co.uk.

Reference	Description	1	2	3	4	5	6	7
	PCB	*	*	*	*	*	*	*
	EP2C5T144C8N Mini Board	*	*	*	*	*	*	*
	DIL IC Socket 14-pin 0.3"							3
	DIL IC Socket 16-pin 0.3"							4
	DIL IC Socket 20-pin 0.3"			1	3			1
	DIL IC Socket 32-pin (or cut-down 40-pin) 0.6"		1					
	DIL IC Socket 28-pin 0.6"						1	
	DIL IC Socket 40-pin 0.6"						1	1
R1, 2, 3, 4	Resistor 220R (all are P:7.62mm)	*	*	*	*	*	*	*
R17, 18, 100, 101	Resistor 10k **(R100, 101 are ECO on backside for REV A PCB)**	*	*	*	*	*	*	*
R102	Resistor 10k **(ECO on backside for REV A PCB)**			*				
R5	Resistor 10k				*	*	*	*
R22	Resistor 10k					*	*	*
R7, 20	Resistor 10k							*
R10, 19	Resistor 33R				*	*	*	*
R21	Resistor 2k2					*	*	*
R9, 12, 13, 14, 15	Resistor 150R							*
R11	Resistor 5k6							*
R16	Resistor 100k							*

R6	Resistor 270k							*
R8	Resistor 560k							*
R23	Resistor 47R (PRI_BLUE)	*	*	*	*	*	*	*
R24	Resistor 47R (PRI_GREEN)	*	*	*	*	*	*	*
R25	Resistor 47R (PRI_RED)	*	*	*	*	*	*	*
R26	Resistor 47R (SEC_BLUE)	*	*	*	*	*	*	*
R27	Resistor 47R (SEC_GREEN)	*	*	*	*	*	*	*
R28	Resistor 47R (SEC_RED)	*	*	*	*	*	*	*
RV1	Trim pot 10k RE06690							*
RV2	Trim pot 50k RE06697							*
D1, 2, 3, 4	LED Red, 5.0mm diameter SC08044	*	*	*	*	*	*	*
D5, 6	Diode, Zener, 3v6 SC15559	*	*	*	*	*	*	*
D7	Diode 1N4148 SC07296							*
C2, 3, 32	Capacitor electrolytic 10u radial (D:5mm P:2.5mm) CA07272	*	*	*	*	*	*	*
C20, 29	Capacitor electrolytic 10u radial (D:5mm P:2.5mm) CA07272				*	*	*	*
C4, 6, 7, 11, 17, 19, 22, 30	Capacitor 0.1u disc; decoupler (P:2.5mm) CA08726	*	*	*	*	*	*	*
C1, 5, 9, 13, 14, 16, 18, 21, 23, 26, 27, 28, 31, 33	Capacitor 0.1u disc; decoupler (P:2.5mm) CA08726				*	*	*	*
C24	0.22uF CA06947							*
C15	Capacitor 33pF CA06302							*
C12	Capacitor electrolytic tantalum 1uF (D4.5mm P:5mm) CA05696							*
C10	Capacitor electrolytic tantalum 10uF (D4.5mm P:5mm) CA05684							*
C8	Capacitor electrolytic tantalum 47uF (D4.5mm P:5mm) CA05687							*
C25	Trim cap 5-65pF CA08450							*
U7	Allied Semi AS6C4008-55PCN 32-pin DIP		*	*	*	*	*	*
U9	Allied Semi AS6C4008-55PCN 32-pin DIP		(*)	(*)	(*)	(*)	(*)	(*)
U2	SN74LVC245 20-pin DIP. Accept no substitute.				*	*	*	*
U6	SN74LVC244 20-pin DIP. Accept no substitute			*	*	*	*	*
U5, 16	SN74LVC244 20-pin DIP. Accept no substitute				*	*	*	*
U12	Z80A-PIO (preferably CMOS version) 40-pin DIP						*	*

U1	Z80A-CTC (preferably CMOS version) 28-pin DIP						*	*
U13	WD2797 40-pin DIP							*
U14	SN74LS174 16-pin DIP							*
U17	SN74LS38 14-pin DIP							*
U11	SN74LS367 16-pin DIP							*
U3, 10	SN74HCT123 16-pin DIP							*
U15	SN74LS06 14-pin DIP							*
U8	SN74HCT244 20-pin DIP SC16609				*	*	*	*
U4	SN74HCT10 14-pin DIP							*
J6, 7, 8, 9	14x2, 0.1” pitch female header (for FPGA card)	*	*	*	*	*	*	*
J11	16-way 2x8 male pin IDC connector, polarised CN17032	(*)	(*)	(*)	(*)	(*)	(*)	(*)
J4	16-way 2x8 male pin IDC connector, polarised CN17032						*	*
PL2	16-way 2x8 male pin IDC connector, polarised CN17032			*	*	*	*	*
PL3	16-way 2x8 male pin IDC connector, polarised CN17032					*	*	*
J20	34-way 2x17 male pin IDC connector, polarised CN16434							*
J5, J10	2x2, 0.1” pitch male header (or wire link)							(*)
J12	1x3, 0.1” pitch male header (or wire link)	(*)	(*)	(*)	(*)	(*)	(*)	(*)
J3	1x2, 0.1” pitch male header (or wire link)							(*)
J1,2,14,15,16	1x4, 0.1” pitch male header	(*)	(*)	(*)	(*)	(*)	(*)	(*)
PL4	26-way 2x13 male pin IDC connector, polarised CN16432						*	*
SW1, 2, 3	PCB Momentary Tactile Push Button Switch Right Angle With stent 6*6*5mm (search Aliexpress)	*	*	*	*	*	*	*
J17, 18	DB15 female high-density VGA connector There are at least 2 designs of this connector available, with different footprints. This PCB requires sockets where the rows of pins are on a 2.0mm pitch, *not* a 2.54mm pitch.	*	*	*	*	*	*	*
J13	Mini Din 6 female PS/2 keyboard connector	*	*	*	*	*	*	*
J19	micro SD card socket on breakout board		*	*	*	*	*	*

For interfacing you may also need a ribbon cable terminated at each end with a 16-way connector (CN21933) and/or one terminated at each end with a 26-way connector (CN21936).

Construction notes for stage 1:

Fit/solder R23-28. These control the color of the monochrome display on the primary and secondary VGA outputs. Fitting all 6 resistors will give a white display. Fitting R24, 27 will give a green display. By choosing a suitable combination of values you should be able to get an amber display (if you do, let me know the values because I have not experimented with this..).

Fit/solder all of the remaining resistors (or, all of the resistors for this stage).

Fit/solder the two zener diodes D5, 6, observing polarity: match the stripe on the diode with the stripe/K-marking on the silk screen.

Fit a jumper wire to link pins 1-2 of J12.

Fit/solder the 0.1uF decoupling capacitors (best to do all of these during stage 1).

No ICs are fitted for stage 1. However, if you plan to fully populate the board, and intend to use IC sockets, fit them all now: it's easier to fit them before any high-profile components are fitted.

Fit J6,7,8,9 – the connectors for the FPGA daughter-card. Use a fine-bladed hacksaw to cut the connector strip to length. Hold the connector against the PCB holes to mark the cuts. Don't try to cut *between* a pair of pins, but cut destructively *through* a pair of pins (you can pull the 2 pins out first if you wish, using a pair of fine pliers). Place all 4 connectors in position on the PCB and gently push the FPGA daughter-card into position: not all the way, but about half-way down. This ensures that all of the connectors are vertical. Like all of the other components, the connectors mount on the *top* of the PCB so that the FPGA daughter-card is mounted face-down, with its connectors overhanging the right-hand edge of the PCB. Solder the corner pins of each connector, then double check that all of the connectors are mounted flush with the PCB and are correctly aligned with the FPGA daughter-card. Solder all of the remaining pins, then remove the FPGA daughter-card.

Fit/solder the switches SW1,2,3.

Fit/solder the 10uF electrolytic bulk decoupling capacitors, observe polarity: match the + with the (tiny) marking on the silk-screen. The orientation is the same for all of these components (best to do all of these during stage 1, but you only need C2, 3, 32 for now).

Fit/solder the LEDs D1,2,3,4, observe polarity: match the flat edge of the LED with the marking on the silk-screen. The orientation is the same for all of these components. *You might want to mount these with the leads bent at 90^{o} so that they are aligned with the switches*.

Fit/solder the VGA connectors J17,18 (or just J17) – solder two of the signal pins then check that the socket is square and flush on the PCB before soldering the lugs.

Fit/solder the PS/2 keyboard connector, J13. **On a REV A PCB this must be mounted on the underside of the PCB. The revision is marked in the top-side silk-screen near to mounting hole H3.**

Fit/solder the header strips J1, 2, 13, 14, 16 (optional – fit as-and-when needed).

On a REV A PCB, implement the modifications described in REV A ECOs on page 52.

Finally, fit a 10K resistor between pins 9 and 10 of socket U2. This provides a pull-down on D7 which effectively disables monitoring of the NASCOM keyboard which would be connected to PL3. You can remove this if you fit U2 for a later stage of assembly.

This concludes stage 1 assembly (Minimum system). Refer to Section Testing on page 46. Construction notes continue for stage 2.

Fit/solder the microSD daughter-card, J19. Place the daughter-card flush on the main PCB and put short pieces of wire through the first and last hole, bending them top and bottom to keep them in place. Solder top and bottom. Check that the daughter-card is flush and square, then put wires through the remaining holes and solder top and bottom.

Fit/solder U9 and (optionally) U7. *I recommend fitting DIL sockets for all of the ICs.*

This concludes stage 2 assembly (SDcard and external memory). Refer to Section Testing on page 46. Construction notes continue for stage 3.

Fit/solder U6.

Fit/solder PL2. In preference, use a socket with a frame and a polarity cut-out; observe polarity: match the cut-out with the marking on the silk-screen.

This concludes stage 3 assembly (Serial port). Refer to Section Testing on page 46. Construction notes continue for stage 4.

Fit/solder the remaining resistors if you omitted them earlier.

Fit/solder U2, 5, 16 (U5 is immediately below C2: the silk-screen marking for U5 is obscured by the capacitor and, because it is vertically slightly higher than the marking for U4 it seems as though the numbering is out-of-order).

Fit/solder U8 (U8 is used as a “bus-keeper” for the I/O bus even if no NASCOM keyboard is fitted, thus it is required for this assembly stage; refer to Section The I/O bus on page 14).

This concludes stage 4 assembly (I/O bus). Refer to Section Testing on page 46. Construction notes continue for stage 5.

Fit/solder PL3. In preference, use a socket with a frame and a polarity cut-out; observe polarity: match the cut-out with the marking on the silk-screen.

This concludes stage 5 assembly (NASCOM keyboard). Refer to Section Testing on page 46. Construction notes continue for stage 6.

Fit/solder U1 and/or U12 (Z80-CTC and Z80-PIO).

Fit/solder J4 and/or PL4 (J4 for the CTC, PL4 for the PIO). In preference, use a socket with a frame and a polarity cut-out; observe polarity: match the cut-out with the marking on the silk-screen.

This concludes stage 6 assembly (PIO and/or CTC). Refer to Section Testing on page 46. Construction notes continue for stage 7.

Fit/solder U3, 4, 10, 11, 13, 14, 15, 17.

Fit/solder the remaining resistors if not already fitted: R6, 7, 8, 9, 12, 13, 14, 15, 11, 16, 20.

Fit/solder C8, 10, 12. These are all electrolytics so observe the correct polarity.

Fit/solder C15, 24.

Fit/solder D7 observing polarity: match the stripe on the diode with the stripe/K-marking on the silk screen.

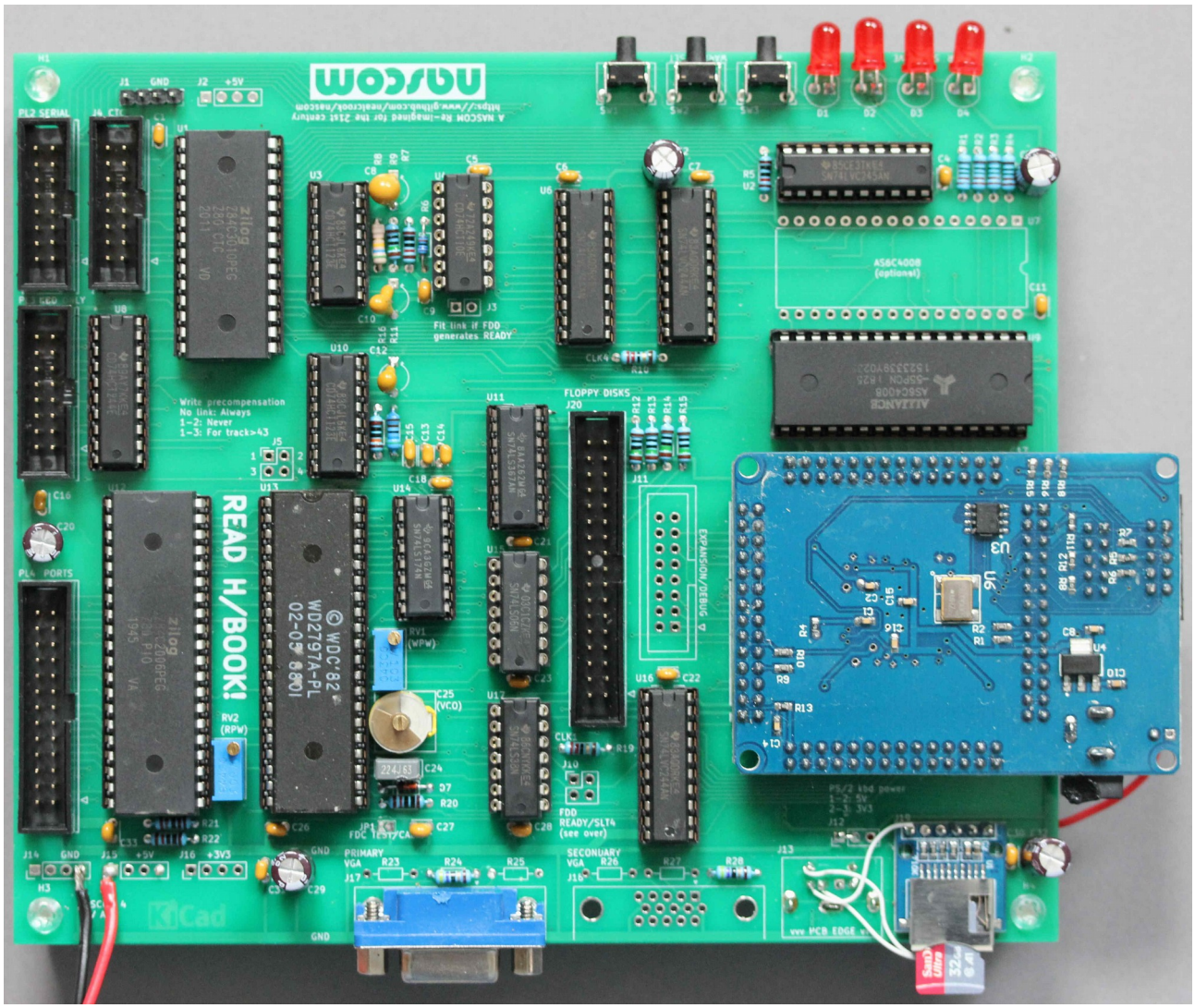
Fit/solder the trimmer capacitor, C25.

Fit/solder the two variable resistors, RV1, 2.

Fit/solder the header strips J5, 10, 3.

Fit/solder J20. In preference, use a socket with a frame and a polarity cut-out; observe polarity: match the cut-out with the marking on the silk-screen.

This completes assembly. Refer to Section Testing on page 46.



Testing

Testing is described stage-by-stage.

Stage 1 (Minimal system)

Program the FPGA (See Programming the FPGA on page 49) and connect it to the PCB if it's not already connected. No other ICs should be fitted for this test.

Connect a VGA monitor and PS/2 keyboard.

Make a 5V connection from the FPGA board to the NASCOM 4 PCB, then apply 5V (see Power on page 7).

The POWER and SDACTIVE LEDS (D4, D3) should turn on and stay on.

All of the LEDs on the keyboard should turn on for about 1s, then turn off so that just CAPS LOCK is on. Pressing the CAPS LOCK key repeatedly should toggle the CAPS LOCK LED.

The screen should be blank. Press SW2 (WARM RESET). The NAS-SYS 3 prompt should appear, with a flashing cursor on the next line.

Key-presses on the keyboard should echo to the screen – including cursor keys and the GRA and CTRL keys (see PS/2 Keyboard on page 7 for key mappings).

Try using T to inspect memory and M to modify memory.

Store a NOP and then HALT, and execute it:

```
MC80
    0C80 00 76.
EC80
```

The HALT LED should turn on and stay on.

Press SW1 (NMI). The HALT LED should turn off and the screen should look like this:

```
1000 F253   0C82 0000   0000 0031   0000 0031
0000 0031   0000 0031   00 0000 0000
```

Type these commands:

```
O EE 0
O EF 0
```

After you type the first command, the screen should go blank for about 2s while the monitor resyncs. The screen should refresh as an 80-column, 25-line display, filled with NULL characters (empty rectangular box) with a flashing cursor under the top-left character. You need to type the second command "blind". After you type the second command, the screen should go blank for about 2s again, then should refresh as the NASCOM 48-column, 16-line display, with its contents uncorrupted.

If everything is working as expected, carry on playing with your minimal NASCOM 4, or go on to test stage 2.

If anything does not behave as expected, check the power and all of the connections. If you can't find/fix the problem, consult the author. Do not proceed with any of the other testing steps.

Stage 2 (SDcard and external memory)

Start with the system connected up as described in the previous stage.

Power down the system. Fit U9 (1st memory device) and insert a programmed SDcard (See Error: Reference source not found on page Error: Reference source not found). Power on the system.

The POWER and SDACTIVE LEDS (D4, D3) should turn on. The SDACTIVE LED should turn off after about 0.5s, and a boot menu should appear on the screen.

A menu item is selected by pressing an upper-case letter, A-Z. Try different menu items. Press SW3 (RESET) to return to the menu.

After selecting a menu item, press SW2 (WARM RESET) to reset the NASCOM 4 into the same environment.

Select menu item H to load images of NAS-SYS 3, BASIC etc. from SDcard to RAM. Use the J command from NAS-SYS to start Microsoft 8k BASIC.

The menu includes a memory test. Select it then type O 18 19 <ENTER>E C80 8000 0 8<ENTER>. See Memory Test on page 11.

If everything is working as expected, carry on playing with your NASCOM 4 system, or go on to test stage 3.

Fault-finding

If the menu does not appear or the system hangs, the problem must be with the SDcard connections, the SDcard image, the external memory or its connections. Press SW2 (WARM RESET) to start up the system using the FPGA ROM version of NAS-SYS 3. If necessary, remove the SDcard and press SW3 (RESET) then SW2 (WARM RESET).

From NAS-SYS 3, use the M and T commands to modify memory at $1000 and then examine it. This should show any obvious data bit errors. Modify several adjacent locations to look for addressing errors.

TODO steps for testing SDcard interface

Stage 3 (Serial port)

Start with the system connected up as described in the previous stage.

Power down the system. Fit U6 (level shifter). Power on the system. See SERIAL connector on page 8 for ways to connect and test the serial interface (O 1D 18 to set up for nascom_sdcard).

If everything is working as expected, carry on playing with your NASCOM 4 system, or go on to test stage 4.

Stage 4 (I/O bus)

Start with the system connected up as described in the previous stage.

Power down the system. Fit U2, 5, 16, 8. Power on the system.

There are no specific tests for this new functionality, except to make sure that everything that was working before carries on working!

If everything is working as expected, go on to test stage 5.

Stage 5 (NASCOM keyboard)

Start with the system connected up as described in the previous stage.

There are no new active components added for this step; the only additional component is the NASCOM keyboard connector. Connect a genuine NASCOM keyboard to PL3. If you are using a NASCOM 1 keyboard, you will need to construct an appropriate cable. Power on the system.

All of the keys should perform their expected function. The NASCOM keyboard and the PS/2 keyboard should work interchangeably.

Power down the system. Disconnect the PS/2 keyboard. Power on the system. It should start up as usual and it should be bootable and usable from the NASCOM keyboard. The RS (reset) button on the keyboard should function as expected

TODO Does it work benignly when the keyboard is absent, or is some kind of jumper/option needed? Also need to try this on a minimum system.

If everything is working as expected, go on to test stage 6.

Fault-finding

Assuming that the keyboard itself is known-good, any problems could be in the cabling, the chips/connections on the PCB or (on a REVA PCB) the implementation of the ECO.

On my system, before adding the ECO pullup resistor on the RESET line, certain keys would trigger a reset. On examination, these were all associated with S4 and S5 outputs of the keyboard, which are either side of the RESET line in the ribbon cable.

Stage 6 (PIO and/or CTC)

Start with the system connected up as described in the previous stage.

Power down the system. Fit U12 (PIO) and/or U1 (CTC) Power on the system.

If the PIO is fitted, use this simple command sequence from NAS-SYS. It puts ports A and B into output mode then writes and reads the data registers:

```
O 6 f  -- Mode 0 (Output)
O 7 f  -- Mode 0 (Output)
O 4 aa
O 5 55
Q 4    -- expect AA
Q 5    -- expect 55
```

If the CTC is fitted, use this simple command sequence from NAS-SYS to confirm that the registers are accessible:

TODO test sequence for CTC

TODO more thorough test (eg, loopback wires and a test program, for each device)

Stage 7 (Floppy disk controller) – including calibration

The floppy disk controller has been tested using a Gotek with flashfloppy software and using Pertec FD250 5.25” floppy disks.

The Gotek only requires +5V power and the +5V and 0V connections can be taken from the appropriate connectors on the main PCB.

5.25” floppy drives require +5V and +12V power and so some other power supply will be needed.

FDC Calibration

The “test” pin must be pulled low after the FDC has been reset. Therefore, *with the board powered up*, use solder to bridge JP1 and then follow all three of the calibration procedures without resetting or powering down the board:

- Monitor WD (Pin 31) and adjust RV1 to get the required write pulse width (200ns)
- Monitor TG23 (pin 29) and adjust RV2 to get the required read pulse width (450ns)
- Monitor DIRC (pin 16) and adjust C25 to get a frequency of 250kHz.

After completing the procedure, power down the board and unbridge JP1.

Using 8” disk drives

The calibration procedure assumes use of double-density 5.25” disks. The controller design has not been tested on 8” disks, but should be usable. An FPGA modification is required to change the FDC clock from 1MHz to 2MHz (this could be made soft-switchable). Before calibration, a write is needed to port $E4 to set EIGHTINCH (bit 5) and /DDEN (bit 4) to the correct values. Please contact me if you want to try an 8” drive.

Programming the FPGA

The FPGA board contains a serial EEPROM. At powerup/reset, the contents of the EEPROM are shifted into the FPGA to provide its configuration. There are two ways to reprogram the FPGA:

- Reprogram the EEPROM. This provides a new configuration that will take effect on the next reset and is non-volatile across power cycles
- Shift the new configuration directly into the FPGA. This provides a new configuration that will take effect on the next reset and is maintained until the next power-cycle.

The same programming dongle is used for both of these methods but the programming file and the board connector are different for each of them.

The FPGA board can be programmed while it is separated from or attached to the NASCOM 4 board. The programming dongle supplies power. However, it probably cannot supply enough power for a fully-populated NASCOM 4 board so it is best to keep the main power connected while programming.

FPGA software

If you simply want to program the FPGA with pre-build fuse files supplied by me, you can probably use any version of the tool-chain. To rebuild the VHDL code from source you need version 13.0.1 (version 13, service pack 1). This is a free download from Intel for Windows or Linux:

https://fpgasoftware.intel.com/13.0sp1/

This is a relatively old release (2013). I run it on Linux without any problems but there is a bit of finessing required on modern distributions in order to get the modelsim simulator working; contact me if you have trouble and I will document the flow.

Reprogramming the EEPROM

- Connect the programmer to the AS connector on the FPGA daughter-card (the connector closest to the edge of the board)
- Start the “programmer” application/tool
- In the programmer window, selectrt Mode: “Active Serial Programming”
- Click “Change file” and navigate to NASCOM4.pof
- Select boxes “Program/Configure” and “Verify”
- Press “Start”. Programming/verification takes about 5s.

After programming, disconnect the cable (the board is held reset while the cable is connected).

Shifting configuration data directly into the FPGA

TODO

Programming service

Contact the author for free FPGA programming (you to pay postage costs in both directions).

Power supply

TODO

Mounting the PCB in a case

The PCB has mounting holes in each corner (marked H1-4). The holes are 3.2mm diameter for an M3 bolt. The PCB measures 170mm x 145mm and the FPGA daughter-card overhangs by 18mm. The holes centres are inboard by 5mm in each direction and so they are on the corners of a rectangle measuring 160mm x 135mm.

REV A PCB

The first release of the PCB is designated “REV A”. The revision is marked in the top-side silk-screen near to mounting hole H3.

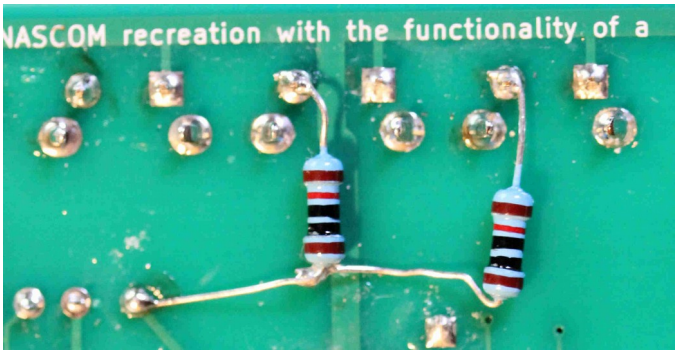
Known bugs/imperfections:

- Silk-screen marking for U5 is obscured by C2 (and its vertical position makes it seem out-of-order with respect to U6).
- The silk-screen does not show the part type for U8.
- The wiring to the PS/2 keyboard connector is mirror-imaged.
- The WARM_RESET and NMI switches require the addition of pull-up resistors because the input-only pins of the FPGA cannot be configured to have internal pullups
- The RESET switch requires a pull-up resistor/RC filter because the internal pullup isn’t strong enough (only a problem when an external NASCOM keyboard is fitted, because the long cable to the reset switch on the keyboard acts as an aerial and picks up cable cross-talk and noise that causes spurious resets or resets when certain keys are pressed).
- On PL2, the RX data needs a pullup resistor to stop the UART from “dribbling” incoming NULL characters.
- On J7 (FPGA connector) pins 11 and 12 are swapped (the 50MHz clock is on pin 12)
- The control signals for the NASCOM keyboard are wired wrongly; KBDCLK and KBDRST go through the level shifter as though they are outputs from the keyboard/inputs to the FPGA.

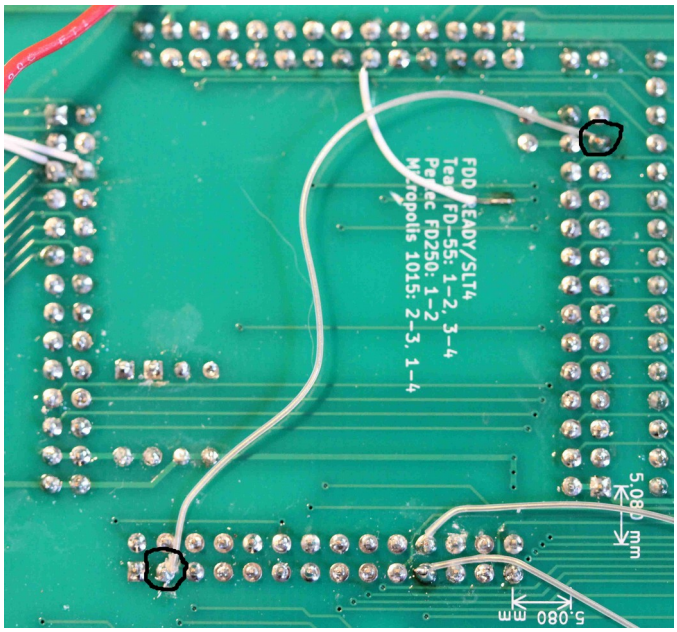
REV A ECOs

The following modifications are required on REV A PCBs.

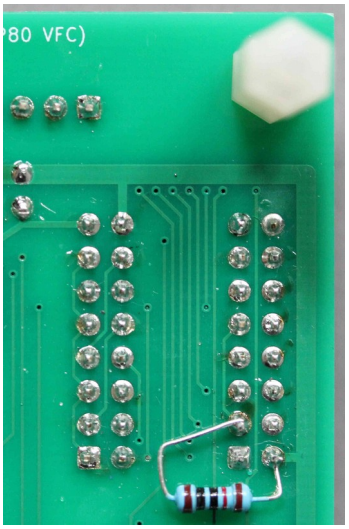
1. Mount the PS/2 keyboard socket, J13, on the **underside** of the board; this compensates for the mirror-image wiring.
2. Fit 10K pullup resistors (R100, R101) from the WARM RESET and NMI switches to 3V3 (2 resistors). These can be fitted on the back of the PCB and connected to 3V3 at a nearby pad. See photo.



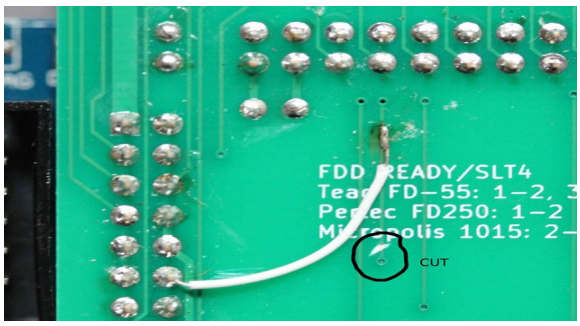
3. Add a filtering RC network to the RESET switch (this network exists on the FPGA card so this ECO simply requires a wire-add from J6/25 to J8/3. See photo.



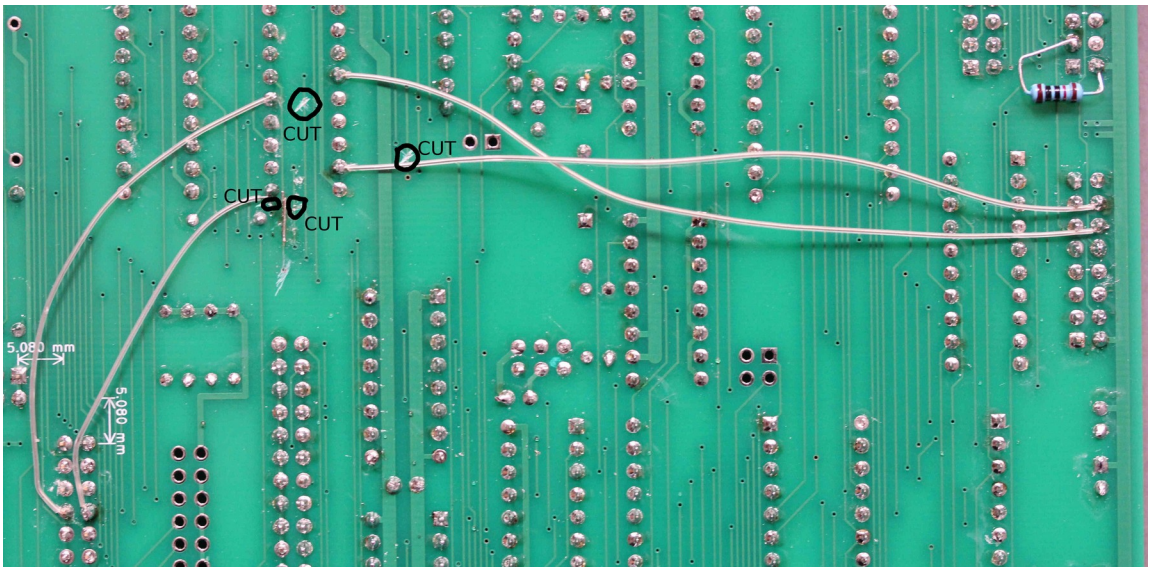
4. Fit a 10K pullup resistor (R102) from RX_5V to 5V. This can be fitted on the back of the PCB between pins 2 and 3 of the serial connector, PL2. See photo.

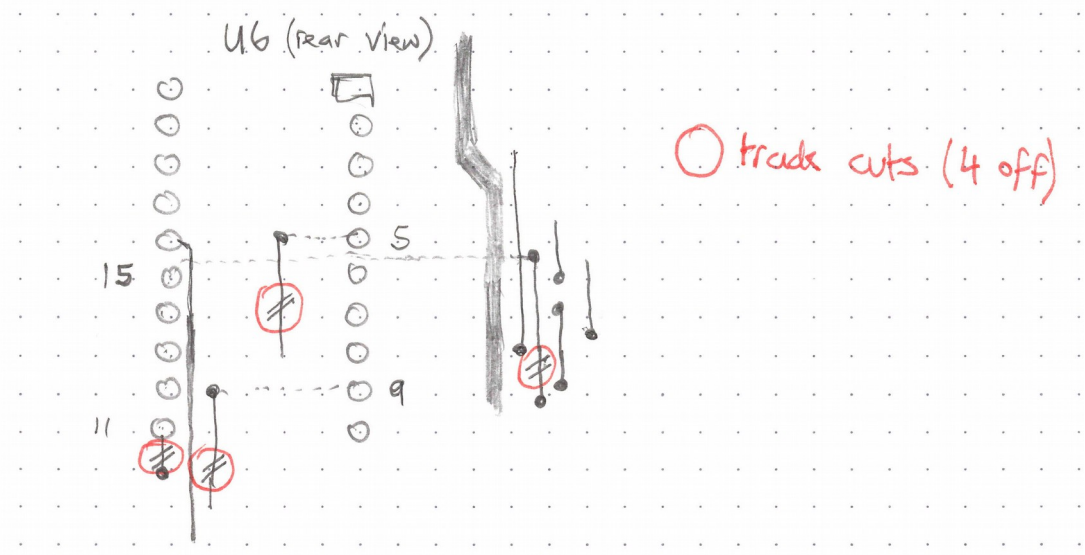


5. Cut the connection from J7 pin 11 and wire PIN18_IN_WARMRST to J7 pin 12: use a scalpel to scrape the solder resist from the track. See photo.



6. Make 4 cuts to disconnect U6 pins 5, 9, 11, 15. Add 4 wires to re-route those signals: (1) Wire from J8 pin 22 to U6 pin 15. (2) Wire from J8 pin 21 to U6 pin 11. (3) Wire from U6 pin 5 to PL3 pin 12. (4) Wire from U6 pin 9 to PL3 pin 14. See photo and sketch.





Acknowledgements

To my beloved Father for lending me the extra £100 or so to top-up my saving enough to buy my NASCOM 2 + RAM A card. My Brother drove us to Compshop in New Barnet in his Morris Marina. The date was 15th April 1980 and the price (including VAT) was £339.25 including a free 3A power supply. As I told Dad years later: “that was the best investment you ever made in my education” (and yes, I did pay him back).

To all contributors to INMC magazine and its successors, for teaching me so much about the NASCOM.

To the organisers and members of the NASCOM Thames Valley User Group (NAS-TUG) for friendship and knowledge-sharing.

(fast-forward a few decades)

To Grant Searle for his multicomp design which is the spiritual ancestor of this design as well as the origin of the VGA video sub-system, the PS/2 keyboard interface and the UART.

To the members of https://groups.io/g/Nascom-Computers for their interest, encouragement and support.

To all the folk who develop and maintain KiCad.

To all the developers of the T80 Z80-core.

Hardware Releases

There is one PCB revision released by me. It is marked REV A on the top-side close to one of the corner mounting holes. This revision requires various cuts/wires (see REV A ECOs on page 52).

There are various releases of the “menu” that is loaded from the SDcard. Early releases did not include the ability to use the SDcard as storage for PolyDos or CP/M.

These are the releases of the bitstream programmed into the FPGA EEPROM. The bitstream defines the behaviour of the FPGA hardware, including the built-in ROMS: NAS-SYS 3, VFC, Character generator, SBROM.

Rev1.0 – All the “REV A” PCBs assembled/pre-programmed by me used this version. There’s no visible version number on the hardware to identify it.

Rev1.1 – From this version onwards, the FPGA/SBROM version is identified on the top line of the NASCOM screen at boot. The changes include support of simultaneous NASCOM and VFC video output and the programmable character generator.

Rev1.2 – Only FPGA change is support of the INVVIDEO bit. Associated menu release adds nasForth.

Rev1.3 – FPGA change is to modify .qsf file to add pull-ups to some inputs so that a minimal system + SDcard + memory works correctly without the need to terminate unused inputs.

Change History

20Feb2021	Neal Crook	1st Edit.
02Mar2021	Neal Crook	First draft release
22Mar2021	Neal Crook	Updates after PCB bringup.
02Apr2021	Neal Crook	Fix duplicate C13 in parts list. Add polarity bit to SERCON. Describe how to use nascom_sdcard serial interface. Add most of the bring-up guide.
11Apr2021	Neal Crook	Describe ECO. Add photos
17Apr2021	Neal Crook	Details on $EE/$EF assignment. Notes on running memory test and on connecting disk drives. Document MC6845 registers.
19Apr2021	Neal Crook	Tidy up SDcard register descriptions. More on PS/2 keyboard keys. Revise FDC setup procedure.
02May2021	Neal Crook	Revise serial comms stuff. Change ECO description to add RC network to RESET line. Added details/status on different DOS.
23May2021	Neal Crook	Describe utilities for Local SDcard version of PolyDos.
09Oct2021	Neal Crook	Fix typos. Rename make_sdcard_image to make_rom_menu, Describe Local SDcard version of CP/M, Update to show latest boot menu. Re-organise material about SDcard generation, describe sdcard_editor and other scripts.
20Nov2021	Neal Crook	Add section to desctibe video subsystem, including programmable character generator. Add control bit to port18 description. Add “Hardware Releases” section with release history.
18Apr2022	Neal Crook	Describe Port$EC reset state and INVVIDEO bit. Describe RAM used by FPGA.
13Jun2022	Neal Crook	Describe Rev1.2 release.
03Dec2022	Neal Crook	Revise construction section to encompass REV B PCB: add resistors 100,101,102 which were ECOs on REV A. Clarify polarity of electrolytics. Describe release 1.3