NASCOM 4 Handbook

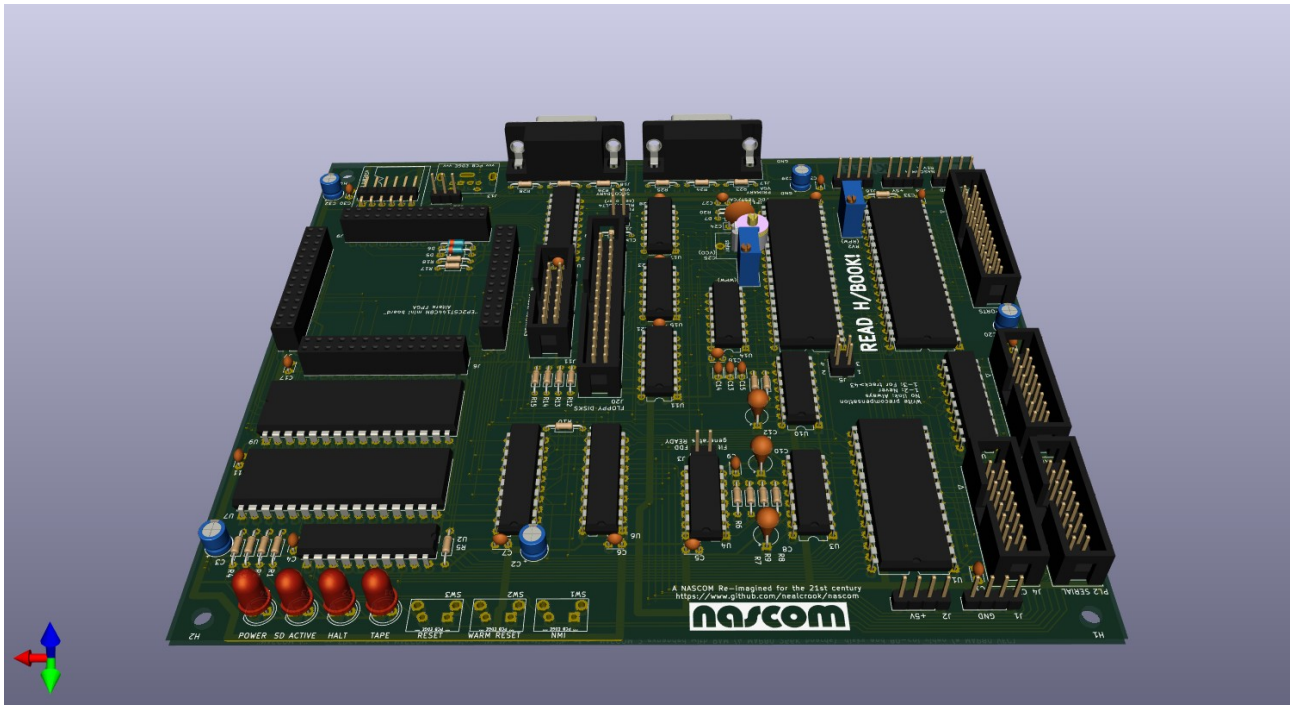


Table of Contents

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 “SDcard active”
- Push-buttons for “Reset”, “NMI” and “Warm Reset” (warm reset is explained later)
- 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 microSD socket. The microSD card holds ROM images that can be loaded automatically at reset.

Implementation and build options

The main “guts” of the design are implemented within an ALTERA FGPA. 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.

A “minimum build” requires the FPGA board, connectors, buttons and LEDs but no other active components. It provides most of the capabilities of an unexpanded NASCOM 2:

- Z80 CPU
- NAS-SYS 3
- 48x16 memory-mapped video, with NAS-CHAR and NAS-GRA character sets, driving VGA output
- NMI single-step
- Input through PS/2 keyboard
- 6402 UART
- 1 Kbyte RAM (Workspace RAM)

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 microSD card 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.
- Add a handful of 74-seried 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.

A guide to the board

The board uses 2 power domains: +5V and +3V3. 5V Power is supplied to the FPGA daughter-card through a barrel connector (check polarity carefully) and is regulated there to generate 3V3 for the FPGA itself.

3V3 is supplied to the board from the FPGA regulator output through connectors J6, 7, 8, 9. These connectors also provide the 0V reference. (TODO I hope there is enough spare current; there should be: there was for multicomp but now we have some assorted 74LVC logic). A flying lead from the daughter-card to the main PCB supplies +5V.

The power grids are rather haphazard (I'd do that differently another time) 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.

The two VGA connectors should drive any normal LCD. If you find a monitor that does not behave, let me know. At the moment I am aware that the final pixel of the NASCOM screen seems to be truncated and I will investigate a solution for that at some point.

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

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

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 NASCAS board, or to a USB-to-serial adaptor.

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 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. I expect to provide guidance later on setting this up for a GOTEK unit as well as some real drives.

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.

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 VCO, the TODO and the TODO.

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 connectors!! TODO

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 your local charity shops or freecycle 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.

1 Maybe that wasn’t a brilliant idea..

Some keys 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.

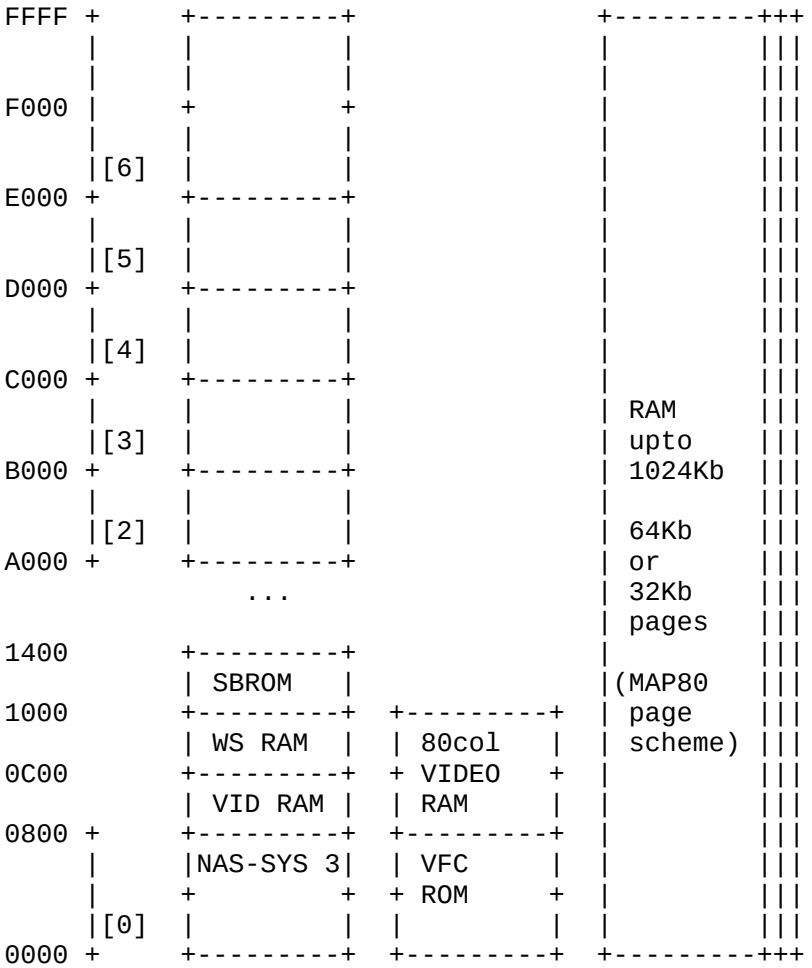
TODO photo

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 18).

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
$E0-$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) NOT IMPLEMENTED		MAP80 VFC
$EB	(VFC MC6845 data) NOT IMPLEMENTED		MAP80 VFC
$EC	VFC memory mapping	FPGA	MAP80 VFC
$EE	VFC select VFC video on output	FPGA	MAP80 VFC
$EF	VFC select NASCOM video on 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 but fixed at 1 on NASCOM 4. Receive buffering is 1 byte on NASCOM 1/2 but 16 bytes on NASCOM 4.

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
0			Unused: read 0.

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).

Port: $02 (READ-ONLY)
Name: UART STATUS
Notes: Read-only UART status.

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
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 17).

Port: $11 (READ)
Name: SDSTATUS
Notes: SDcard status register. Refer to SDcard Interface (page 17).

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			Unused: read 0.

2			Unused: read 0.
1			Unused: read 0.
0			Unused: read 0.

Port: $11 (WRITE)
Name: SDSTATUS
Notes: SDcard status register. Refer to SDcard Interface (page 17).

Bit	Reset	Name	Description
7			Unused: write data ignored.
6			Unused: write data ignored.
5			Unused: write data ignored.
4			Unused: write data ignored.
3			Unused: write data ignored.
2			Unused: write data ignored.
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 17).

Port: $13 (WRITE-ONLY)
Name: SDLBA1
Notes: SDcard address register. Specifies bits [15:8] of the logical block address. Refer to SDcard Interface (page 17).

Port: $14 (WRITE-ONLY)
Name: SDLBA2
Notes: SDcard address register. Specifies bits [23:16] of the logical block address. Refer to SDcard Interface (page 17).

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		Unused: write data ignored, read 0.
5	0	VFCAUTO	MAP VFC autoboot. Determines effect of VFCMAP[1] (Port $EC). This mimics the function of the L4 jumper on the 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 18).
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. These write-protect address regions correspond to the NASCOM monitor ROM, EPROMs on the main NASCOM board and the NASCOM 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 18)

Port: $1C (READ-ONLY)
Name: REASON
Notes: Bit7 distinguishes cold from warm reset (it is the *only* difference between cold and warm reset). TODO additional functionality is planned but not yet implemented.. including write (Any value) to clear reason code.

Bit	Reset	Name	Description
7		WARMRST	0: Cold reset 1: Warm reset
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	0		Unused: read 0.
0	0		Unused: read 0.

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: FM (single), 1: MFM (double)
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: $EC
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	TODO ??
2		INVVIDEO	TODO not implemented .. but could be
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, wrtiting VSOFT=1 *disables* the ROM.
0		VIDRAM	Enable VFC video RAM in high half of 4Kbyte block

Port: $EE
Name: VFCVIDVFC
Notes: A 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)

Port: $00 (WRITE)
Name: VFCVIDNAS
Notes: A 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=$80 (ensures previous command has completed)
- Write the block address to LBA2, LBA1, LBA0 (you only need to write values that have changed since the last command. For example, if LBA2 remains at 0, there is no need to write it again)
- Write 0 to SDCTRL to issue the READ command
- Loop 512 times: wait until SDSTATUS=$E0 (read byte ready, block busy), read byte from SDDATA.

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
- Loop 512 times: wait until SDSTATUS=$A0 (ready for write byte, block busy), write byte to SDDATA.

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 a code example, refer to the subroutine SDRD512 in the SBROM source code.

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 SBootROM:

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

Reset (cold and warm) and boot

There are four sources of reset:

- Power-on reset
- Reset from the cold 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

- 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 the REASON register (port $1C) has bit[7] set for a warm reset. 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.

In the case of 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 high byte of the destination is written to the PORPAGE port.

In the case of a warm reset, 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.

TODO need to check for and handle the case where there is NO SDCARD fitted. Display message and wait for keypress which takes you to “default config”

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 (TODO currently upper case only, but maybe nice to fix that). 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.

SDcard format

The SDcard has no file-system structure imposed on it; it is accessed as a sequence of 512-byte blocks. The script make_sdcard_image has been created to facilitate the creation of this image.

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

Menu Examples

The make_sdcard_image script (written in PERL) creates the SDcard image in the correct format. 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: + POLYDOS3\r" .
    "K  NAS-SYS3 + BASIC + NASDOS\r" .
    "L: NAS-SYS3 + PASCAL\r" .
    "M: NASCOM CP/M\r" .
    "N: MAP80 VFC CP/M\r" .
    "O: NAS-SYS3 + LOLLIPOP\r" .
    "P: NAS-SYS3 + INVADERS\r" .
    "Q: NAS-SYS3 + NASPEN\r" .
    "R: 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 nascas 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");
# NAS-SYS1 in RAM
add_profile('D', "L:NASSYS1",                               "P19=01", "G0=11");
# NAS-SYS3 in RAM
add_profile('E', "L:NASSYS3",                               "P19=01", "G0=11");
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:POLYDOS3",  "L:BASIC",    "P19=60", "GD000=19");
add_profile('K',              "L:NASDOS",    "L:BASIC",    "P19=60", "GD000=19");
add_profile('L',                             "L:PASCAL",   "P19=60", "GD000=19");
# No protect
add_profile('M', "L:NASCPM",                                          "GF000=3");
# MAP80 CP/M – with video switch
add_profile('N',                                            "PEE=00", "G0=20");
add_profile('O',               "L:LOLLIPOP",                          "G1000=19");
add_profile('P',               "L:INVADERS",                          "G1000=19");
add_profile('Q', "L:NASPEN",                                "P19=08", "G0=19");
# Load then start NAS-SYS
add_profile('R',               "L:MEMTEST",                           "G0=19");
```

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_sdcard_image, the symbolic name (eg ZEAP) created by add_image() provides the image size, location on SDcard and load address.

When executed, make_sdcard_image 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 I86 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 I70 LD000=10 I34 LE000=10 P19=60 GD000=19
INFO profile L as I58 LD000=18 P19=60 GD000=19
INFO profile M as I80 LF000=4 GF000=3
INFO profile N as PEE=00 G0=20
INFO profile O as I84 L1000=2 G1000=19
INFO profile P as I8E L1000=7 G1000=19
INFO profile Q as I4C LB800=4 P19=08 G0=19
INFO profile R as I95 LC80=2 G0=19
```

Looking at one profile in detail (profile H):

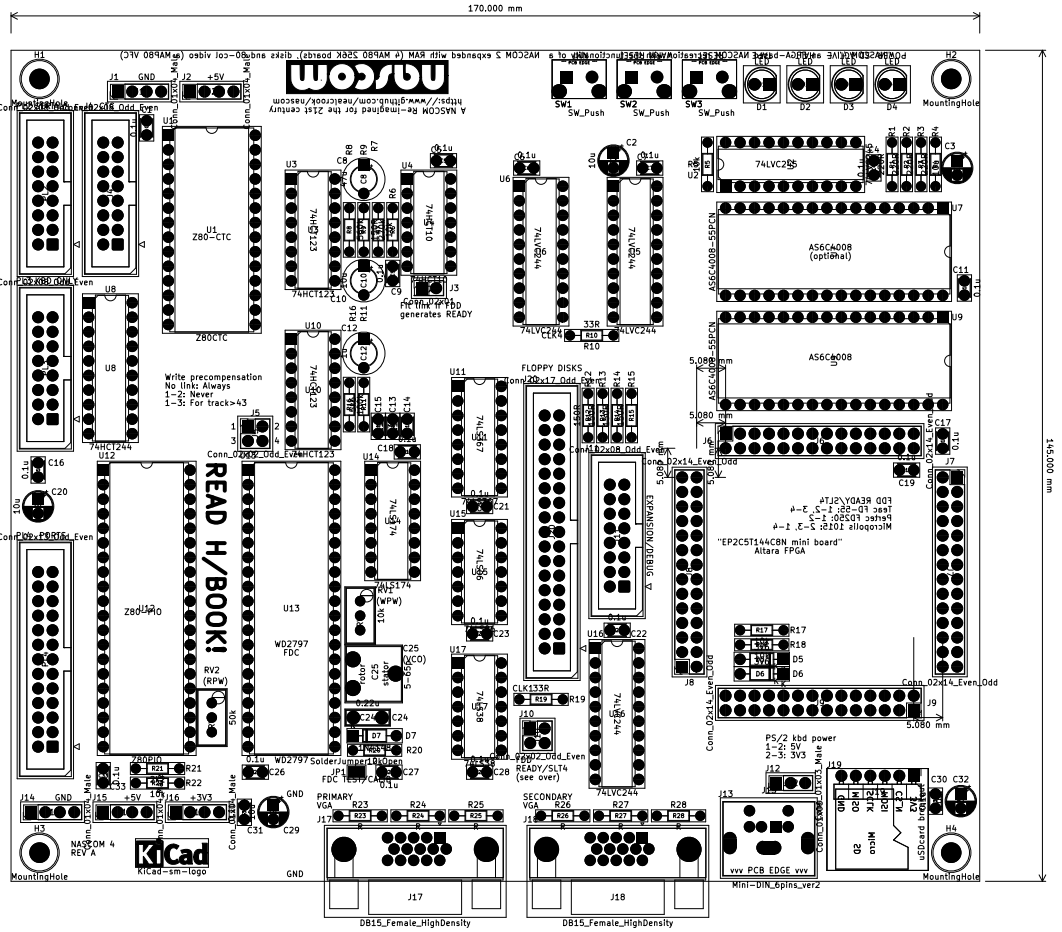
```
I86 LC000=8 I44 LD000=8 I34 LE000=10 P19=70 G0=19
```

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 10) 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.



Construction

The PCB is double-sided with through-hole plating. The boards are 100% tested at manufacture. All of the 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.

I recommend that you construct your board in stages, testing it after each stage. The stages are:

1. Minimal system
2. SDcard and external memory

3. I/O bus (the stages after this all require the I/O bus)
4. KBD connector
5. PIO and/or CTC
6. 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
	PCB	*	*	*	*	*	*
	EP2C5T144C8N Mini Board	*	*	*	*	*	*
R1, 2, 3, 4	Resistor 220R (all are P:7.62mm)	*	*	*	*	*	*
R5	Resistor 10k			*	*	*	*
R10, 19	Resistor 33R			*	*	*	*
R21	Resistor 2k2				*	*	*
R22	Resistor 10k				*	*	*
R9, 12, 13, 14, 15	Resistor 150R						*
R7, 20	Resistor 10k						*
R11	Resistor 5k6						*
R16	Resistor 100k						*
R6	Resistor 270k						*
R8	Resistor 560k						*
R23	Resistor TBD (PRI_BLUE)	*	*	*	*	*	*
R24	Resistor TBD (PRI_GREEN)	*	*	*	*	*	*
R25	Resistor TBD (PRI_RED)	*	*	*	*	*	*
R26	Resistor TBD (SEC_BLUE)	*	*	*	*	*	*
R27	Resistor TBD (SEC_GREEN)	*	*	*	*	*	*
R28	Resistor TBD (SEC_RED)	*	*	*	*	*	*
R17, 18	Resistor 10k	*	*	*	*	*	*
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	*	*	*	*	*	*
D7	Diode 1N4148						*
C2, 3, 32	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	*	*	*	*	*	*
C20, 29	Capacitor electrolytic 10u radial (D:5mm P:2.5mm) CA07272			*	*	*	*
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 TODO						*
C15	Capacitor 33pF CA06302						*
C13	Capacitor electrolytic tantalum 1uF (D4.5mm P:5mm) CA05696						*
C10	Capacitor electrolytic tantalum 10uF (D4.5mm P:5mm) CA05684						*
C8	47uF TODO						*
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.			*	*	*	*
U5, 6, 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				*		
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					*	
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 (or wire link)	(*)	(*)	(*)	(*)	(*)	(*)
PL4	26-way 2x13 male pin IDC connector, polarised CN16432					*	
PL2	16-way 2x8 male pin IDC connector, polarised CN17032	*					
PL3	16-way 2x8 male pin IDC connector, polarised CN17032				*		
SW1, 2, 3, 4	PCB mount momentary action push	*	*	*	*	*	*
J17, 18	DB15 female high-density VGA connector	*	*	*	*	*	*
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).

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

Fit/solder the 10uF electrolytic bulk decoupling capacitors, observe polarity: match the + with the 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 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 SW1,2,3,4.

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 90º so that they are aligned with the switches*.

Fit/solder the VGA connectors J17,18 (or just J17).

Fit/solder the PS/2 keyboard connector, J13.

Fit/solder the header strips J1, 2, 12, 13, 14, 16.

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 1 assembly. Refer to Section Testing on page 30. 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. Refer to Section Testing on page 30. Construction notes continue for stage 3.

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

Fit/solder U2, 5, 6, 6 (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). As noted above, I recommend fitting DIL sockets for all of the ICs.

This concludes stage 3 assembly. Refer to Section Testing on page 30. Construction notes continue for stage 4.

Fit/solder U8. As noted above, I recommend fitting DIL sockets for all of the ICs.

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 4 assembly. Refer to Section Testing on page 30. Construction notes continue for stage 5 (PIO/CTC).

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 5 assembly. Refer to Section Testing on page 30. Construction notes continue for stage 6 (Floppy-disk controller).

Fit/solder U3, 4, 10, 11, 13, 14, 15, 17. As before, I recommend fitting DIL sockets for all of them.

Fit/solder the remaining resistors: 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 30.

Testing

Testing is described stage-by-stage.

Stage 1 (minimal build)

Stage 2 (SDcard and external memory)

Stage 3 (I/O bus)

Stage 4 (KBD connector)

Stage 5 (PIO and/or CTC)

Stage 6 (Floppy disk controller) – including calibration

Programming

FPGA board, programming dongle, cable, software/version, two different files/ programming service

Programming service

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

Preparing an SDcard

TODO

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 TODOmm. The holes centres inboard 5mm in each direction so that they are 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).

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.

Change History

20Feb2021	Neal Crook	1st Edit.
02Mar2021	Neal Crook	First draft release