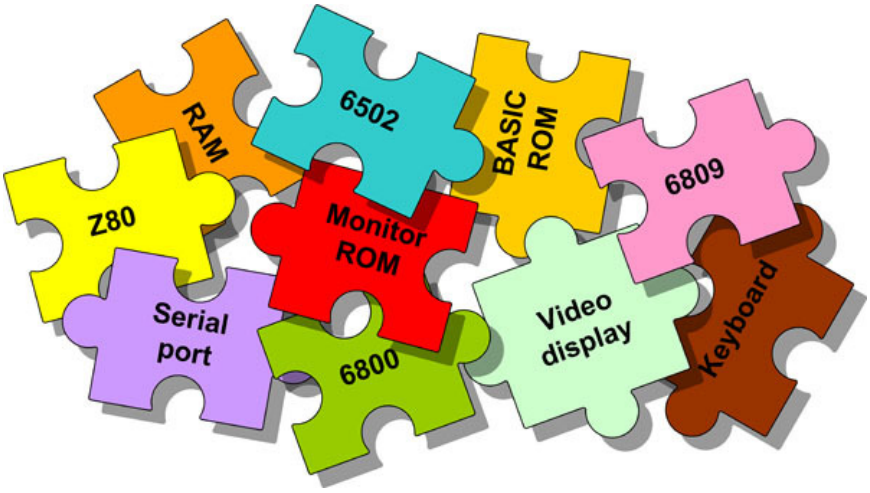
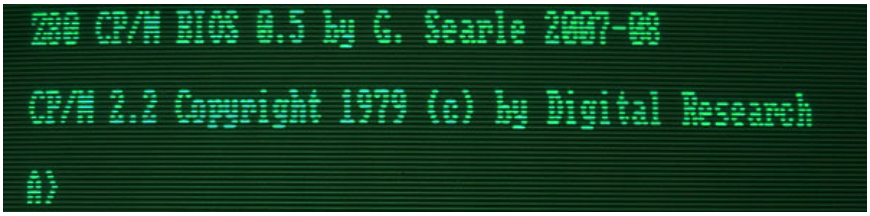


CP/M on FPGA

(An add-on project to my multi-option FPGA design)



PAGE STILL BEING UPDATED REGULARLY
Please come back again (and REFRESH the browser page) to see if there are updates.
IMPORTANT: Due to code updates, please make sure you have latest files before using this page.

VERIFIED WORKING
RE-BUILT FROM INFO PROVIDED HERE

by Grant Searle

For news and updates, follow me on Twitter:
Follow @zx80nut

Last update: 19th March 2014

NOTE: This is an add-on project to the main build HERE and is adapted from my discrete hardware design page HERE.

Major changes log:
** 17th February 2014 - Initial release*
** 19th March 2014 - Correction to formatter program (not all directory entries were being wiped) - many thanks to Oscar Vermeulen for spotting and correcting the issue*

Contents

Introduction
Building the machine
Using the machine
How to install CP/M
How to install applications
CP/M system files (HEX and ASM)
Memory maps
Disk details
IOBYTE details
ROM BASIC details

Terminal screenshots
Links

INTRODUCTION

This project shows how it is possible to construct a fast Z80 computer running full CP/M 2.2 with disk storage using a low cost FPGA board. Source code and is included to allow modifications if needed. Several commercial CP/M packages have been tried on this and they work perfectly. A large library of applications are available elsewhere on the web, and these can be loaded onto this machine using my own custom-made transfer software.
Full usage instructions are included on this page.
Also included is a ROM version of BASIC to allow the machine to be programmed without CP/M.

Same as for my original CP/M project (here) a LOT of work has gone into this, as I have not only designed the hardware, but written the Boot ROM, BIOS, Formatting, System Transfer and File transfer programs. Also, many hours spent in making the ROM BASIC as clean as possible, with extended commands written by myself within it.

Features
CPU: Z80 core running at up to 25 MHz
Interface: Two high-speed serial ports at 115200 Baud or one serial port, one monitor/TV out and one PS/2 keyboard.
Disk: SD Card (NOT SDHC), 128MB utilised, containing 8 or 16 logical drives, respectively.
RAM: 64K Byte
ROM: 8K Byte, switched off when CP/M active. ROM contains the bootloader and memory load utilities. Also contains Microsoft BASIC, as used on the NASCOM computer, modified to remove code that is not relevant to a serial-interfaced board (eg. screen handling and keyboard matrix scanning).
Resets: Both cold (full reset) and warm reset (used to return to CP/M prompt) circuitry
Power consumption: Less than 250mA
CP/M support: 2.2 with included software.

The ROM is turned off by writing to port 38H.

BUILDING THE MACHINE

Firstly, ensure you have the LATEST FILES that are on the main page. You will need to use the ROM that I have provided to allow the CP/M to be booted, and this has only been included in the latest ZIP files.

I COULD just provide the complete VHDL, however, the intention with this and the main page is to allow you to see what is needed and build accordingly. Therefore I will show you how to change what you would do for a standard Z80 machine to make it CP/M - enabled.

This is an enhancement to a Z80 machine that is to be built as shown on the main FPGA multi-computer here.

Proceed with that page and create a machine with the following requirements:
Z80 Processor
External FULL RAM
Interface 1: Serial port
Interface 2: Serial port OR Monitor/keyboard port
SD Card
Clock speed can be anything up to the maximum 25MHz.

Compile the machine and test that it works properly.

Once working, come back here and make the following changes to the VHDL:

Step 1. Replace the ROM with the ROM BASIC and CP/M boot file

change

```
rom1 : entity work.Z80_BASIC_ROM -- 8KB BASIC
port map(
address => cpuAddress(12 downto 0),
clock => clk,
q => basRomData
);
```

to

```
rom1 : entity work.Z80_CPM_BASIC_ROM -- 8KB BASIC and CP/M boot
port map(
address => cpuAddress(12 downto 0),
clock => clk,
q => basRomData
);
```

Step 2. Enable ROM page-out.
When CP/M starts, the ROM is to be disabled. The CP/M images are set up so that a write to $38 will turn off the ROM, resulting in ALL of the 64K address space being the SRAM instead.

To do this, a new signal is needed that is used to turn off the ROM (and re-activate it if the board is reset)

New code shown in red, existing code in blue...

```
  :
signal serialClock : std_logic;
signal sdClock : std_logic;

--CPM
signal n_RomActive : std_logic := '0';

begin

--CPM
-- Disable ROM if out 38. Re-enable when (asynchronous) reset pressed
process (n_ioWR, n_reset) begin
if (n_reset = '0') then
n_RomActive <= '0';
elsif (rising_edge(n_ioWR)) then
if cpuAddress(7 downto 0) = "00111000" then -- $38
n_RomActive <= '1';
end if;
end if;
end process;

  :
```

The update the chip select code to ensure the ROM can only be selected if the new active signal is enabled (active LOW)...

Change
`n_basRomCS <= '0' when cpuAddress(15 downto 13) = "000" else '1'; --8K at bottom of memory`
to be
`n_basRomCS <= '0' when cpuAddress(15 downto 13) = "000" and n_RomActive = '0' else '1'; --8K at bottom of memory`

These are the only changes required. Wire up the board as shown on the main multi-computer page with a 2GB or 1GB SD (or microSD) card connected.

Compile and upload. The machine is then ready to use.

USING THE MACHINE

Connect the board to a 5V regulated supply capable of supplying at least 250mA (or 500mA if you have a PC keyboard connected).

The board is set up for 115200 Baud, hardware handshake, 1 stop bit and no parity.
Connect it to a PC or similar running a terminal program set to these parameters, then press the reset button.

```
Press [SPACE] to activate console
```

...is displayed on both serial ports (or serial port and tv/monitor if designed that way). Only one is active at any time, so a spacebar press on the terminal/keyboard will allow the board to determine which port is to be used for the console I/O.

Once spacebar is pressed, the following is displayed...

```
CP/M Boot ROM 2.0 by G. Searle
BC or BW - ROM BASIC Cold/Warm
X        - Boot CP/M (load $D000-$FFFF)
:nnnn... - Load Intel-Hex file record
Gnnnn    - Run loc nnnn

>
```

BC
This starts the Microsoft BASIC ROM interpreter in "Cold" mode (ie. clears memory)

You will see

```
Memory top?
(press ENTER for full RAM usage)

Z80 BASIC Ver 4.7b
Copyright (C) 1978 by Microsoft
```

```
52755 Bytes free
Ok
```

BASIC is then ready for programming

BW
This starts the Microsoft BASIC ROM interpreter in "Warm" mode (ie. doesn't clear memory, so a program already in memory will remain)

X
Boots CP/M
You will see

```
Boot CP/M?
```

displayed. Press "Y" to allow CP/M to be loaded from the disk. This will also turn off the boot ROM, allowing full 64K RAM access to CP/M.
If a properly initialised and CP/M-installed disk is present (see below for instructions), the following is then displayed...

```
Loading CP/M...
screen clears here, so you may not see the above line
Z80 CP/M BIOS 2.0 by G. Searle 2013

CP/M 2.2 Copyright 1979 (c) by Digital Research

A>
```

CP/M is then loaded and ready to use.

INSTALLING CP/M

Installing CP/M onto a new card is a straightforward process, and involved firstly formatting the card then transferring the CP/M system to the first sectors on the card.
The board will need to be connected to a PC (or whatever) which has a terminal program and contains the ".HEX" files needed for the installation.
Reset the machine, and press spacebar, as described above. The machine is then ready to load Intel-Hex files if they are transferred from the terminal.
The following steps require Intel-Hex files to be loaded. To do this, open the ".HEX" file in something like notepad, select the whole contents (CTRL-A) then "Copy" to clipboard. In the terminal, paste the contents into the window - the board will automatically load them to the appropriate locations specified in the HEX file. Dots will be shown on the terminal for each byte loaded. Once the end of the hex file is reached, "File loaded" will be displayed.
If for any reason something fails, simply cold-reset the machine and repeat the steps.

PART 1. FORMATTING THE DRIVE

Load the Intel-HEX dump of FORMAT into memory by copying the contents of FORM128.HEX (for 128MB SD utilisation) into the terminal window.
FORMAT resides at $5000 when loaded.
Executing address $5000 will start the FORMAT program which will then initialise the directory structures (ie. a quick format, as that is all that is needed on an SD card) for ALL of the logical drives on the flash card.

Once loaded, type **G5000** then press ENTER.

You will see...

```
CP/M Formatter by G. Searle 2012
Then the following will appear in sequence (only takes a few seconds)
ABCDEFGHIJKLMNOP
Finally
Formatting complete

>
```

Each drive is 8MB so 128MB on the card will have drives A: to P:

PART 2. "PUTSYS"

CP/M is "installed" on the first track of a disk. When the disk is then booted, the first track is read into the correct area of memory then executed.
To write the CP/M system to the disk, it is firstly loaded into memory, then a program is executed to write the memory to the first sectors on the flash disk.

To install, the following steps are needed in this order:
1. Load the Intel-HEX dump of CP/M into memory by copying the contents of CPM22.HEX into the terminal window. This will take about 10 seconds to load.
2. Load the Intel-HEX dump of CBIOS into memory by copying the contents of CBIOS128.HEX (for 128MB total drive space) into the terminal window. This takes about 3 seconds to load.
3. Load the Intel-HEX dump of PUTSYS into memory by copying the contents of PUTSYS.HEX into the terminal window.

PUTSYS resides at $5000 when loaded.
Executing address $5000 will start the PUTSYS program which will copy the CP/M and CBIOS from memory to the flash drive.
Type **G5000** then press ENTER.

You will see...

```
CP/M System Transfer by G. Searle 2012

System transfer complete

>
```

Once these are complete, CP/M is then installed on the disk and can be booted. Applications will then need to be installed.

INSTALLING APPLICATIONS

To install files, a small client application needs to be installed on the A: drive on the board. Once this is done, the windows program to package the files is used to create a text file which is then pasted into the terminal window.

PART 1 - The CP/M client application installation

The installation of the client application only needs to be done once (provided it is not deleted) on each flash card.

1. Install CP/M as shown above then boot from CP/M (press X and then Y at the monitor prompt, as described above).
2. Press the cold reset button to get back to monitor
3. Paste DOWNLOAD2.HEX (a relocated DOWNLOAD.HEX) to terminal window. This will load the download client into high memory. This normally resides at 0100, but as the ROM is active, this area is not available.
4. Type **GFFE8** and press ENTER (this will relocate RAM 4100-->0100 and restart CP/M)
6. When in CP/M, **SAVE 2 DOWNLOAD.COM** and press ENTER

A full terminal session of this is shown here...

```
<<<PRESS COLD RESET>>>
Press [SPACE] to activate console
<<<PRESS SPACE>>>
CP/M Boot ROM 2.0 by G. Searle
BC or BW - ROM BASIC Cold/Warm
X        - Boot CP/M (load $D000-$FFFF)
:nnnn... - Load Intel-Hex file record
Gnnnn    - Run loc nnnn

>X <=== Type char
Boot CP/M?
<<<PRESS Y>>>
Loading CP/M...
Z80 CP/M BIOS 2.0 by G. Searle 2013

CP/M 2.2 Copyright 1979 (c) by Digital Research

A>
<<<PRESS COLD RESET>>>
Press [SPACE] to activate console
<<<PRESS SPACE>>>
Z80 Monitor
Type ? for options
<<<PASTE THE DOWNLOAD2.HEX FILE INTO THE TERMINAL WINDOW>>>
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
>................................
```

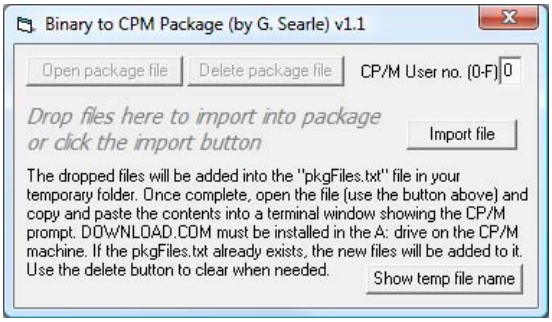
```
>Load complete.

>GFFE8   <=== Type chars and press ENTER
Screen clears here
Z80 CP/M BIOS 2.0 by G. Searle 2013

CP/M 2.2 Copyright 1979 (c) by Digital Research

A>SAVE 2 DOWNLOAD.COM    <=== Type chars and press ENTER
A>
```

DOWNLOAD.COM is now present on the A: drive, and the machine is now ready to be loaded with application files. There is no need to repeat the above steps unless the DOWNLOAD.COM file is deleted from the A: drive.

PART 2 - Using the Windows packager program

This can be used as many times as required and can transfer single or multiple files into the current drive that is active in the terminal window.

This is a small Microsoft Windows application that will allow BINARY CP/M files to be dropped onto it, which will then add them to a package text file. If you prefer, you can use the "import" button to select a file instead of dropping files onto the window.



NOTE: The "pkgFiles.txt" file will be created in the temporary folder of your machine. Only limited error handling has been added to this program, so if it fails please re-run.
Once all files that are to be transferred into a particular drive have been dropped, the package file can then be opened.
Copy the COMPLETE contents of this file and paste it into the terminal window of the machine.
The terminal must be showing the CP/M prompt. This can be on any drive. The packaged files will be created in the current drive on the CP/M machine.
The "DOWNLOAD.COM" file on A: drive on the CP/M board will then start and read the HEX byte stream, storing it in a file.
Multiple files are permitted in a package file, allowing you to transfer a complete application containing many files in a single operation.
Partial contents of this file are, as an example...

```
A:DOWNLOAD BBCBASIC.COM   <==== Command to create a file BBCBASIC.COM in the current drive
U0  <==== User selection
:C33401C37B01C3A501C3CA01C37A01C37301C38501C37201DDE5F   <==== Data stream, starts with a colon
CB5FD21003CCA120FC5E5CDD209E1CDD50ACC9209C179B7CA990   <==== Data stream
36F7E0>0085   <==== Data stream, ends with a greater-than symbol, followed by a checksum
A:DOWNLOAD STAT.COM   <==== Command to create second file STAT.COM in the current drive
U0  <==== User selection
:C33304202020436F707972696768742024576363456E2342EEA2   <==== Data stream for second file, starts with a colon
B0E0ECD4C14C9212A15702B712A2915EB0E015CD3805C6414F   <==== Data stream
A1A1A1A1A1A1A1A>80DD   <==== Data stream, ends with a greater-than symbol, followed by a checksum
```

An example terminal session showing these files being loaded is shown here...

```
Z80 CP/M BIOS 2.0 by G. Searle 2013

CP/M 2.2 Copyright 1979 (c) by Digital Research

A>C:  <=== Select drive where files are to be loaded and press ENTER
<<<PASTE CONTENTS OF PKGFILES.TXT>>>
C>A:DOWNLOAD BBCBASIC.COM <== Start of pasted text
.............................................   <=== Import progress of first file
.............................................
OK   <=== Successful import of a file completed
C>
C>A:DOWNLOAD STAT.COM
.....................................   <=== Import progress of second file
OK   <=== Successful import of second file completed
C>
C>DIR    <=== A DIR command will show the created files
C: BBCBASIC COM : STAT COM
C>BBCBASIC   <=== Running one of the imported files
BBC BASIC (Z80) Version 3.00
(C) Copyright R.T.Russell 1987
```

You can, of course, install your own transfer program using the above then use that.

The format of my transfer files is simple so you can also create your own program to build the transfer text files if you don't want to use my Windows app.
You need to create a text file with the following contents (also refer to my example above)

First line:
***A:DOWNLOAD <filename>** where <filename> is the CPM file that is required. Must be a space between DOWNLOAD and the filename.*
Second line:
***U0** This is the "user" number that can see the destination file. If you are not familiar with CP/M users, then always use U0*
Third line:
A continual stream of HEX values** (must be 2 chars each) for each byte in the file, followed by a **>** character, followed by the **checksum

Each line is terminated with a carriage return / linefeed ($0D, $0A) as standard for text files

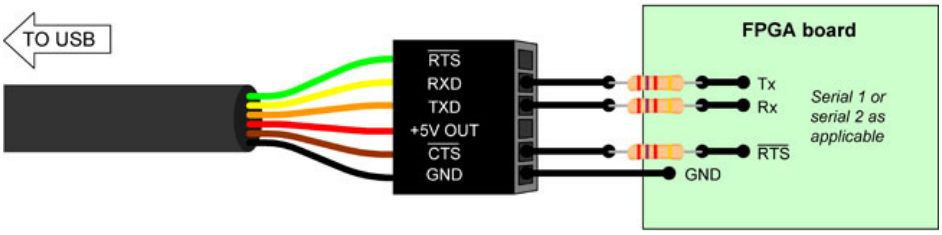
The checksum is two pairs of HEX values:
The first one is the low-byte of the length of the file being uploaded. Because CP/M files are normally saved in blocks of 128, this is normally "80" or "00", but doesn't need to be.
The second one is the low-byte of the SUM of each byte being uploaded.
The checksum is very simple, but more than adequate for confirming a simple data transfer, as it can detect missing, extra or changed characters.

If multiple files are to be transferred, then all can be included in the same text file immediately after one another, with each following the three line format shown above.

Suggested serial connection

This can be connected to a USB to TTL (3.3V) serial cable as shown below.

Using a USB to 3.3/5V (TTL) serial cable instead of an RS232 interface



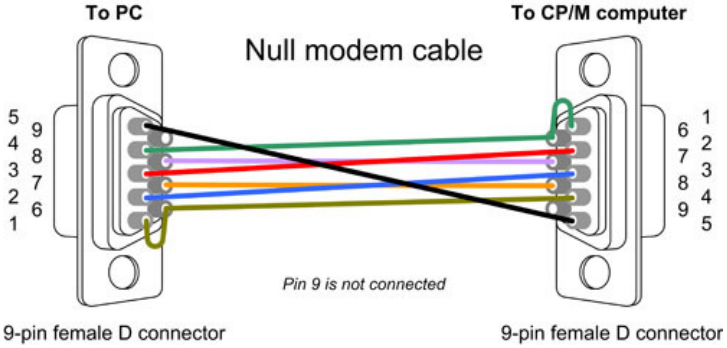
Resistors are present to safely limit current along the cable if the USB or computer is not powered.

By using a USB to serial cable the board can also be powered from the USB port down the same cable (a powered hub could be used if higher current needed), eliminating the need for a separate power supply for this board. Just connect the +5V out on the cable to the centre pin of the 5V input supply on the board.

The 2k7 resistors are present in the diagram because the board may be powered but not plugged in to the USB cable, or the USB cable could be plugged in and active without power to the board. These limit the current to avoid power being drawn through the interface pins. If always powering the board via the cable then no resistors are necessary.

OR

If using an RS232 converter then a suitable cable needs to be made to connect the CP/M machine to a PC to allow the system to be transferred from the files supplied below. Here is a fully configured null modem cable that I use successfully:



CP/M SYSTEM FILES

DOWNLOAD FILES HERE

Note: 19th March 2014 - Correction to formatter program (not all directory entries were being wiped) - many thanks to Oscar Vermeulen for spotting and correcting the issue.

NOTE: THE ROM BOOT file is provided with the main FPGA multi-computer.

The ZIP file presented here contains the following files. Please read the rest of the page to see what files are needed and when to use them.

Windows file packager
FilePackage.exe
COMDLG32.OCX <== *Copy to the same folder as FilePackage.exe if you get an error saying it is missing when running the packager program.*

CP/M "standard" transient programs
CPM211FilesPkg.txt
Optional, but available pre-packaged ready to install if needed. See further down as to how to use the package files (basically, open it and paste all into a terminal window showing the CP/M prompt after the download program has been installed on the CP/M A: drive). Contains LOAD, PIP, STAT, DDT, DISPLAY, DUMP, ED and ASM, as supplied with CP/M 2.2.

CP/M Installation files
BASIC.HEX
DOWNLOAD2.HEX
MONITOR.HEX
CBIOS128.HEX
CPM22.HEX
FORM128.HEX
PUTSYS.HEX

Source code for the installation files
basic.asm
download.asm
monitor.asm
cbios128.asm
cpm22.asm
form128.asm
putsys.asm

DOS/Windows assembler
_ASSEMBLE.BAT
TASM.EXE
TASM80.TAB
TASM_RELNOTES.TXT
TASM.TXT

Acknowlegements
The BIOS, Format, Putsys and the file transfer programs are my own work.
CPM22 is a modified version found on the net. Originator copyright within the file.
Monitor is partially my own work, but using routines written by Joel Owens.
BASIC is Microsoft BASIC 4.7 for the NASCOM, heavily modified by myself to remove references to different monitors, screen handlers and keyboard matrix reading.
TASM assembler is a partial distribution of the package from Speech Technology Incorporated.

MEMORY MAPS

INITIAL BOOT

0000-1FFF ROM (8K)
  Within the ROM...
    Boot monitor followed by the Microsoft BASIC interpreter
2000-FFFF RAM

Once CP/M is loaded

0000-FFFF RAM

0100 Transient program area (applications)
D000 CP/M System
E600 BIOS

DISK DETAILS

Sixteen CP/M drives are available.
A: to P: - 8MByte

The block size was chosen as a compromise between smallest possible, but without using too much RAM (because CP/M stores a complete allocation bitmap of all blocks, so smaller blocks would require more RAM to be allocated to each drive).
So, the chosen block size is 4096 bytes (*).
512 Directory entries for each drive was chosen as being suitable (*).
These can be changed within the source code, but require changes to the BIOS and also to the formatter.
The sizes that I have used have been fine for me, though, and I recommend they remain as-is, unless you are very familiar with the parameter tables.

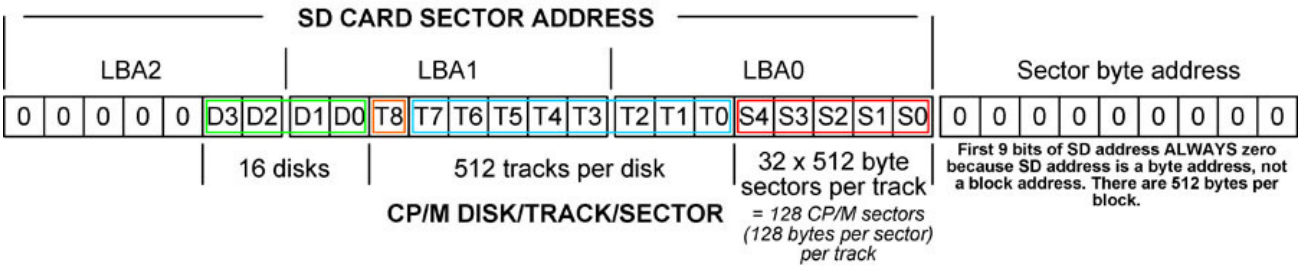
The standard CP/M sector size is 128 bytes, and I have used the published blocking/deblocking algorithms (by Digital Research) to allow the 512 byte sectors to be used (ie. 4 CP/M sectors are contained in each SD card sector).

** - Each directory entry points to up to 8 blocks. For the values that I have chosen each block is 4096 bytes. So, if all files were 4K, a maximum of 2MB on a drive will be used (512 x 4 k files). However, if an average of 16K is assumed (each entry therefore pointing to 4 blocks) then the full 8MB will be used. This is largely academic, however, as you are very unlikely to fill either the disk capacity or directory entries.*
More directory entries would reduce the amount of free RAM available to the user. A smaller block size would also have the same affect.

Sector addressing

LBA addressing is used because not all cards have the same drive geometry (sectors per track). All SD cards support LBA. Using LBA, a contiguous sector address is used to address the complete disk.

The CP/M drive, track and sectors values are bit-shifted and transferred into the LBA addresses (higher LBA address bits set to zero) as shown below...



The shifting, splitting and combining of each of the values can be seen in the CBIOS code ("setLBAaddr").
This is the physical sector address. Each sector is 512 bytes and contains 4 x CP/M sectors (128 bytes each).
The CP/M track size is arbitrary, so 128 sectors per track seems a suitable value to use - this makes 32 "blocked" sectors per track.
To get 8MB disks, this therefore requires 512 tracks, so 9 bits are used for the track number.

IOBYTE details

(If you don't know what the CP/M IOBYTE is then you can safely ignore this section!)

For full details on how to map logical devices to physical devices, please refer to the CP/M 2.2 manual available on the internet.

The CP/M IOBYTE (location 0003 in the memory map) is implemented to allow logical devices to be re-directed to the two physical ports.
The physical ports on this machine have been assigned to "CRT:" (the TTL serial port A) and "TTY:" (the RS232 serial port B).
For flexibility with all logical devices...
the CRT: port (TTL serial) is also assigned to physical devices UP1:, UP2:, UR1: and UR2:
the TTY: port (RS232 serial) is also assigned to physical devices LPT:, UL1:, RDR:, and PUN:

During startup you are asked to press the Spacebar on the window that is being used. This allows the machine to identify which serial port is connected to the user terminal. Depending on which of the serial ports is identified (CRT or TTY) the IOBYTE is set appropriately.

So...
If you activate the terminal on the CRT port, the IOBYTE is set to 00000001b ie. List is TTY:, Punch is TTY:, Reader is TTY:, Console is CRT:
If you activate the terminal on the TTY (RS232) port, the IOBYTE is set to 00000000b ie. List is TTY:, Punch is TTY:, Reader is TTY:, Console is TTY:

These can be changed dynamically as required.

eg. from the CP/M prompt, if STAT is installed...
STAT CON:=CRT: - to set the console to the TTL port
STAT CON:=TTY: - to set the console to the RS232 port
...see the CPM 2.2 manual for more details.
Alternatively, the I/O can be changed within, say, Microsoft CP/M BASIC by "poke"ing the IOBYTE (location 3) with a suitable value. This allows programs to utilise BOTH serial ports for input and output within the same program.

PIP and STAT use these physical and logical mappings. Other communications programs (eg. Kermit) will also work with them.

ROM BASIC (Microsoft BASIC 4.7) - details of what has been included/excluded

Update 8th September 2013 (Boot ROM 1.1) -
1. The HEX and BINARY identifiers have changed to &Hnnnn and &Bnnnn to match Microsoft implementations for other processors.
2. Width setting changed to 255 default, so no auto CR/LF are added when printing long strings.
3. HEX$(nn) and BIN$(nn) now have leading zeroes suppressed, as for other numeric output.
4. CR/LF processing changed slightly internally so that an LF is no longer auto-generated when sending a CR to the terminal.

INCLUDED TOKENS

SGN,INT,ABS,USR,FRE,INP,POS,SQR,RND,LOG,EXP,COS,SIN,TAN,ATN,PEEK,DEEK,LEN,STR$,VAL,ASC,CHR$,LEFT$,RIGHT$,MID$
END,FOR,NEXT,DATA,INPUT,DIM,READ,LET,GOTO,RUN,IF,RESTORE,GOSUB,RETURN,REM,STOP,OUT,ON,NULL,WAIT,
DEF,POKE,DOKE,LINES,CLS,WIDTH,MONITOR,PRINT,CONT,LIST,CLEAR,NEW
TAB,TO,FN,SPC,THEN,NOT,STEP
+,-,*,/,^,AND,OR,>,<,=

Note: there is also SET,RESET,POINT that call user-defined entry points, as in the ORIGINAL Nascom ROM (ie. not changed). Don't use unless you have defined the calling points for them (see the assembly listing for details).

EXCLUDED TOKENS (don't do anything if called)

SCREEN,CLOAD,CSAVE

NEW (my additional implementations)

HEX$(nn) - convert a SIGNED integer (-32768 to +32767) to a string containing the hex value
BIN$(nn) - convert a SIGNED integer (-32768 to +32767) to a string containing the binary value
&Hnn - interpret the value after the &H as a HEX value (signed 16 bit)
&Bnn - interpret the value after the &B as a BINARY value (signed 16 bit)

IMPORTANT NOTES

Integers in this version of BASIC are SIGNED - ie. -32768 to +32767. This includes memory locations when peek, poke, deek,doke commands are issued (etc.)
So, to refer to location "n" above 32767, you must provide the 2's compliment number instead (ie."n-65536") otherwise you will get an "?FC Error" message.

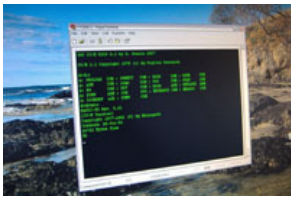
Functions that return integer values (such as the memory FRE(0) are also **signed** (!)) so anything bigger than 32767 will appear as a negative value.

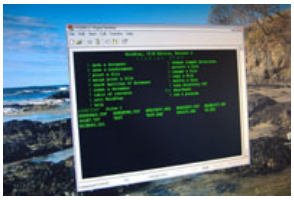
TERMINAL SCREENSHOTS

Taken from my discrete hardware page, but the FPGA version presented here is the same.

(Click on the images to see enlargements)

Here is this machine running on a monochrome (green) monitor using the monitor interface described later on this page:

 Here is the command prompt of my computer running using a Windows terminal.

 And here is the infamous Wordstar (version 4) running on my machine in a Windows terminal.

LINKS

Some of my pages

INDEX PAGE - Go here for access to all of my pages along with my eMail details
Build your own ZX80 - my page showing you how to build this old micro
  |__ ZX80 to ZX81 conversion - build the NMI generator needed to convert the ZX80 circuit into a ZX81
  |__ ZX80 software - Type in a Space Invaders game into the ZX80
Build your own Jupiter Ace - my page showing you how to build this old micro
Build your own UK101 - my page showing you how to build a greatly simplified version of this old micro
A simple 6809 machine - can't get simpler than this for a machine that can run Microsoft Basic
CP/M with 9 chips - build a fully functional CP/M machine with a minimal chip count and CF card storage
Pong - Pictures of my build of the Atari classic arcade game
Practical Wireless Tele-Tennis - A rebuild of the 1974 articles for this home Pong game
My Machines - My collection of classic 80's micros
My TV games - my collection of "pong" style games from the 70's
My electronics kits - my collection of electronics kits from 60's, 70's and 80's

---

Grant.

Back to main menu