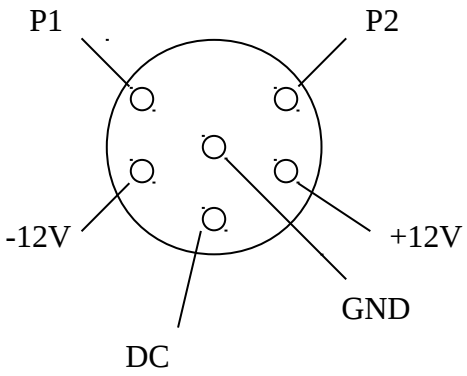
Unofficial Phoenix manual

Work in progress : needs restructuring, etc.

A) Before start up

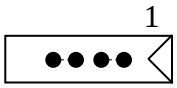
1. Power



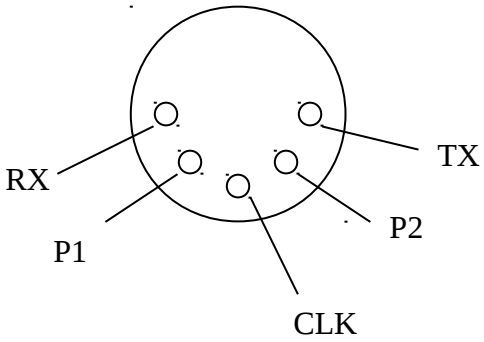
J9 Exterior view

DC is normally sent to a LM323 regulator in order to provide +5V for the whole board…
However it might not be simple to find a power supply that delivers +12V, -12V and a DC between 7.5 and 15V…
So it is surely easier to use a PC PSU that delivers +12V, -12V and +5V. In this case, the DC of the J9 connector has to remain not connected, whilst +5V and GND can be sent to P1 and P2, provided that two wire straps are soldered between P1/P2 and +5V/GND points on the board, because P1 and P2 only cross the board up to the keyboard connector.
This scheme also allows to power the keyboard with +5V/GND, because the other keyboard connector pins are TTL serial signals (see Keyboard section).

Two power connectors are located near the two floppy drives: J6 and J8. An easy way to power the floppy drives is to use two Y power cables with a Molex 5”1/4 connector that remains unconnected, and two female Molex 3”1/2 connectors (one for J6 or J8 connector, the other one for the drive plug).



2. Keyboard



J4 Exterior view

RX, TX and CLK are directly connected to the HD64180 processor's asynchronous serial channel 1, so they are TTL signals.
The rom expects ASCII characters to be sent by the keyboard on this asynchronous serial port and configures channel 1 of the HD64180 with the following format : 19200 baud, 8 data bit, no parity, 1 stop bit. So several options are possible for connecting a keyboard:

a) Using an external computer with a TTL serial interface:

Few computers have a TTL serial interface: mostly single board computers like the Raspberry PI (the TTL serial interface is on the GPIO connector). However any computer with a USB port can be connected with a "USB to TTL-levels serial adapter". Of course a special cable must be built. Please note that the GND reference is not one of J4's pin, except if P1 or P2 is carrying ground from the Phoenix's PSU, so please read the suggestion for connecting a PC PSU…

In this configuration, a terminal emulation software on the external computer will be able to redirect the ASCII characters (typed on the external computer's keyboard) to the Phoenix, but the Phoenix will not send anything to the external computer. With CP/M, it's however possible to use the external computer as a terminal by redirecting the console output to UC1: (the User Channel).
The command to type for that is STAT CON:=UC1:
This is one way to use the Phoenix without a screen, which can be useful if you want to use some software that is not configurable to the specific terminal capabilities of the Phoenix (instead you will use a more standard terminal capabilities with the terminal emulation software of the external computer).

Once again, the Raspberry Pi is an excellent companion to the Phoenix: connected to its keyboard port, it can provide remote access to the Phoenix. My favorite choice is the Raspberry Pi Zero W (Wireless): I supply power to the Pi through the P1 and P2 pins (sending ground and +5V), so the four pins (P1,P2, Tx,Rx) can be directly connected to the GPIO port of the Raspberry, no need for an external power supply. Then I can do a ssh to the Raspberry from anywhere, and from there issue a "screen /dev/ttyS0 19200" command to be connected to the Phoenix. A first press to the Enter key will then be typed in order to pass the "INSERT SYSTEM DISKETTE IN DRIVE A : AND PRESS <RETURN>" prompt. Then the STAT CON:=UC1 command has to be typed blindly, so that the A> prompt will finally appear on the remote terminal.

b) Using a micro-controller driving a keyboard:

Many micro-controllers have a TTL serial interface, so it's easy to connect a micro-controller to the Phoenix' keyboard port, however it has to implement a keyboard driver in order to send ASCII codes to the Phoenix.

c) Using an Oric

It’s possible to connect the parallel port of the Oric to the keyboard port of the Phoenix. For example the Strobe pin of the parallel port can be programmed to emit the bits serially at 19200 baud. Be aware however that the Oric ROM sends glitches on the Strobe pin in two occasions: when powered on, DDRB is initialized before PORTB which makes the Phoenix receive a false start bit, and likewise when the VIA is initialized for K7 operation the Strobe bit is driven low. So, if you don’t want to fix the Oric ROM, the cable between the Oric and the Phoenix must be connected after the Oric has been powered up, and after any program has been loaded from K7.

3. Floppy disk

The Phoenix ROM integrates both the BIOS (Basic Input Output System) and the BDOS (Basic Disk Operating System primitives) of a CP/M 2.2 system. However, it requires a CP/M floppy disk in drive A: in order to load CCP (Console Command Processor), the command line interpreter.

Also, the CP/M disk for the Phoenix have the following format : 10 sectors of 1024 bytes per cylinder, with sectors on side 0 numbered from 1 to 5 and sectors on side 1 numbered from 6 to 10.

Then the disk must have a CP/M file system on it (mainly a disk directory on the first two blocks), and then the CCP has to be extracted from a CP/M 2.2 system and stored in a CCP.COM file, with two additional special checksum bytes (they are checked by the Phoenix’ ROM).

a) Formatting a floppy disk

For now, I recommend using Dave Dunfield’s Imagedisk tool on a PC: it’s not easy to format side 1 of the disk with sectors numbered 6 to 10. Imagedisk uses direct access to a few types of floppy disk controllers, so not all PC floppy controllers are supported: you will have your best luck with old PCs.

First, you need to build the PC-DOS disk of ImageDisk, then you will boot your old PC with it. At boot time, type Enter two times, the distribution will make a ramdisk (E:) with all the programs on it. Type CD \IMD to enter the program directory then type IMD to start ImageDisk.

Now configure some (S)ettings: Sides=2, Double-Step=Off, Interleave=1:1.

Then do a (F)ormat (first time), giving these other parameters: 5 sectors per track, start sector=6 (!), 250 kbps MFM, sector size=1024.

This will format both sides with the special sector numbering from 6 to 10.

Now change again the following (S)etting : Sides=1.

And do another (F)ormat (second time), giving the following parameters: 5 sectors per track, start sector=1, 250 kbps MFM, sector size=1024.

This will format side 0 only, leaving side 1 unmodified, so that you now have sectors numbered from 1 to 5 on side 0 and sectors from 6 to 10 on side 1.

b) Writing a Phoenix disk on a PC

You may use my wphoenix tool (a derivative of writedsk for Oric disks), it will transfer a 800 KB raw image to a real Phoenix disk.

c) How to transfer PC files to a Phoenix disk image

For now, I recommend using the cpmtools on Linux. You will need to compile the distribution and add a new format definition for the Phoenix in diskdefs:

```
diskdef oric_phoenix
  seclen 1024
  tracks 160
  sectrk 5
  blocksize 2048
  maxdir 127
  skew 0
  boottrk 0
  os 2.2
end
```

Then create an empty disk image with

```
dd if=/dev/zero of=disk.dsk bs=1k count=800
```

Now you can initialize a CP/M directory in it with:

```
mkfs.cpm -f oric_phoenix disk.dsk
```

Finally you can transfer files in this disk image with :

```
cpmcp -f oric_phoenix disk.dsk <files> 0:
```

(this will copy files for user 0).

d) Special case of CCP:

Usually, a CP/M computer has its CP/M BIOS in ROM, alongside with a loader that reads the DOS (BDOS) and the command interpreter (CCP) from the first tracks of the disk.

Here the Phoenix' ROM not only contains the BIOS but also the DOS (BDOS primitives). It also has a loader in order to read CCP from disk, but as the BDOS primitives are already present, it's more simple for the loader to load CCP from a file, there's no need to have special tracks at the beginning of the disk.

So, at boot time, the Phoenix loader copies BDOS from ROM to RAM (locations starting at address $E406), then it opens the CCP.COM file on disk A: and transfers the first 2054 bytes to addresses $DC00-$E405. When loading these bytes, a special checksum is calculated, so two additional checksum bytes are loaded and compared with the calculated value (for the normal CCP of CP/M 2.2, these two checksum bytes must be $0F and then $D2).

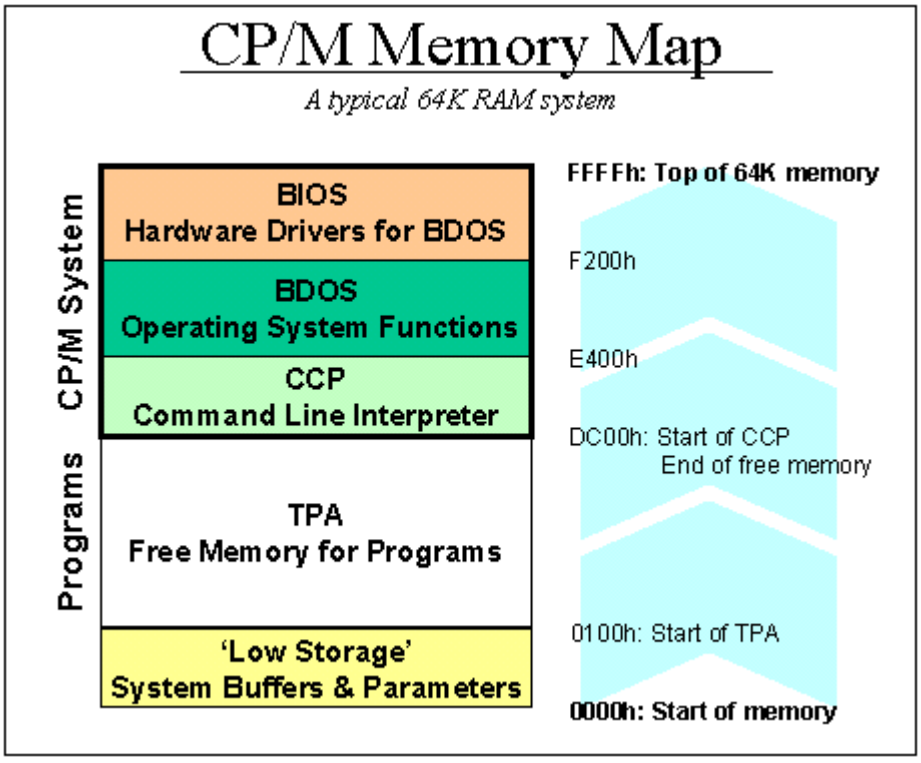
B) Memory usage

256 KB of RAM are present on the Phoenix board, and a second bank of 256 KB can be installed.

Even if CP/M 2.2 normally uses only 64 KB, the 192 additional kilobytes are really useful: the Phoenix manages this ram memory in pages of 16 KB. A first page is used to copy CCP in it, this removes the need to read the floppy disk every time a program terminates (CCP is a transient program: it is loaded in memory range $DC00-$E405 but any executed program can use the full memory space from $0100 to $E405, at the expense of reloading CCP afterwards). On the Phoenix, CCP is thus copied back from this saved page once a program terminates, which is much faster than reading it back from disk. This same page also contains the big printer buffer implemented by the Phoenix BIOS. All the other additional memory pages are linked to form a RAMDISK, so a 176 KB ramdisk is thus initialized on a 256 KB Phoenix.

If you copy the files you need on this ramdisk (M: is the drive letter of this “MemoryDisk”), you will get a very fast computer: for example if you compile source files, especially if compilation requires to write intermediate files… Be careful however to often make backup of the source files, or keep source files on real floppy disks, because any restart of the Phoenix will empty the ramdisk.

The remaining 64 KB are of course those used for programs, here is a memory map usage, showing the separation between the transient program application space, and the BIOS, BDOS and CCP.



(Memory map copied from *obsolescence.wixsite.com/obsolescence/cpm-internals)*

C) Starting CP/M 2.2

Just before running the loaded CCP.COM from disk, the Phoenix inserts a command in CCP’s command buffer. This command is “SUBMIT CONFIG”, which means “execute the contents of the CONFIG.SUB batch file by loading the SUBMIT.COM external command”. If one these files is missing, CCP will complain (for example, “SUBMIT?” if SUBMIT.COM is not on disk), but will still give a prompt and accept interactive commands.

CONFIG.SUB is thus a convenient place to put commands that will be executed at startup, for example it might a good place where to put instructions that will initialize files in the ramdisk…

D) CP/M I/O redirection and BIOS configuration

The Phoenix has a battery-powered Real Time Clock chip that not only aims to keep date and time information (not very useful in CP/M 2.2 as the OS doesn't store timestamps on files), but also BIOS configuration in its on-chip RAM. Up to 50 bytes can be stored in this RAM, and the Phoenix stores the following information:

- format and language of the displayed date (among English/US, French, German, Italian, Spanish, Slovenian and Croatian)… Language is also used to redefine a few accentuated characters when Slovenian or Croatian language is selected.

- configuration of the TTY port: baud rate, parity, stop bits, and whether flow control is used.

- a flag telling if drive B is present (and another one for hard disk, but no hard disk support is given in the BIOS).

- Disk Parameter Block for drive P of CP/M (records per track, block shift factor and mask, extent mask, maximum allocation block number, largest directory number, directory allocation bitmap, size of checksum vector, reserved tracks at beginning) and additional variables (eg. skew scheme).

- CP/M I/O byte.

The I/O byte of CP/M gives redirection for the four logical CONSOLE, READER, PUNCH and LIST devices. The BIOS supplies routines for such I/O redirection, so that it is possible to redirect the four logical CP/M devices (CON:, RDR:, PUN:, LST:) to physical BIOS devices using the STAT command. However, among all the possible physical devices, the Phoenix BIOS only implements TTY:, CRT:, UC1: and LPT: physical devices, ie the other devices are not implemented: Paper Tape Punch (PTP), Paper Tape Reader (PTR), User defined inputs 1 and 2 (UR1 and UR2), User defined outputs 1 and 2 (UP1 and UP2) and User defined Listing 1 (UL1) have no implementation. Also the Batch device (BAT) only has its output defined (redirected to LPT), so if the CONSOLE is redirected to BAT, then the first call for the input status will reset the Phoenix.

This means that the CONSOLE (CON:) can only be redirected to either CRT: (default), TTY: (RS232 port), or UC1: (User Console 1, ie. keyboard port), more on this later. Also, the READER and PUNCH devices (RDR: and PUN:) can only be assigned to TTY: (serial port). And the LIST device can only be assigned to the LPT: physical device.

A small note on UC1:, this serial port for the keyboard has its input and output implemented. This means it is possible to redirect the CONSOLE to UC1 (with the "STAT CON:=UC1:" command), and thus all the characters normally sent to the CRT display will be redirected to the keyboard port. This is great because it allows an easier remote use of the Phoenix than with the TTY serial port.

If you want to connect your own terminal to the TTY port (RS232 connector), you first have to issue a "STAT CON:=TTY:" command on the current CONSOLE device (normally the keyboard and CRT). Then you will be able to interact with the Phoenix through its TTY port. However, even if you save this redirection configuration in the battery backup-ed RAM, when the Phoenix is reset

it will only accept the response to the “INSERT SYSTEM DISKETTE IN DRIVE A : AND PRESS <RETURN>” message from the keyboard, so you will need physical access to the keyboard.

On the other hand, if you connect a TTL-levels terminal to UC1 (the keyboard connector) and issue a “STAT CON:=UC1:” command, then your terminal will be used for the response of the above prompting message. This means you can use the Phoenix as a headless server, accessing it for example from a connected Raspberry Pi, and thus remotely from the Internet.

E) BIOS functions

The BIOS area depicted above (starting at $F200) is copied from the ROM. It starts with a table of jumps that invokes the standard CP/M 2.2 BIOS functions. However, the BIOS for Phoenix was too big to fit in this place so the RAM area for the BIOS mainly contains variables and buffers, and each jump in the BIOS functions table will redirect to the proper function in ROM after having re-activated the ROM in the $0000-$3FFF range.

Details for every BIOS function can be found for example on
https://www.seasip.info/Cpm/bios.html

In the Phoenix’ ROM, there are additional functions not provided by a normal CP/M 2.2. Most of these additional functions exist in a CP/M 3 system, but the Phoenix is not a CP/M 3 system. Unfortunately, these additional functions are not numbered like on a CP/M 3 system, and the usual conventions for using registers are not respected… (this prevents using the Bios function of Turbo Pascal for them).

F) Notes on the implementation of the Console Output function

A big part of the Phoenix ROM is dedicated to the handling of the EF9345 video processor: this is a very complex chip (better known for its use in the Minitel). The BIOS allows to output characters to the screen without having to know anything about this video processor, thanks to a few control characters and ESCAPE sequences. In turn, the BDOS’ console output primitive will use the BIOS functions to write characters to the screen, and every CP/M program will use BDOS’ console output function.

In the two subsections below, the control characters and ESCAPE sequences are listed for the two different screen modes (40 columns and 80 columns), comparing them with the standard they tend to comply with. Additionally, a short list of sequences that don’t fall in these two standards is given here :

```
ESC 0   write to the status line (first line on screen #1 in 80 columns mode)
ESC 1   screen #1
ESC 2   screen #2
ESC 3   screen-saver (empty screen with double-height flashing cursor)
ESC 4   40 columns mode
ESC 8   80 columns mode
```

One can note that they give access to two screens (or pages), each of them can be independently in 40 or 80 columns (however, the status line is only displayed in 80 columns mode on the first page).

40 columns mode : Teletext standard

In 40 columns mode, the EF9345 chip normally allows to define one's own character sets, this feature is used by the ROM which defines a standard ASCII set of 96 characters, but the functionality is not given to the user, he/she will have to resort to assembly programming… But before that, maybe the first missing feature will be that there's no sequence to position the cursor in this mode. Actually, the ROM gives a nearly full implementation of the standard ESCAPE sequences of the broadcast Teletext specification, thus defining serial attributes (like on the Oric!). The EF9345 handles parallel attributes, so it is somewhat under-utilized for this implementation.

With 8 KB of video RAM associated to the video processor, one could also have hoped for using the 4-colors character sets, even in lower resolution, but this feature is not given either by the BIOS, so it will only be possible with assembly programs.

```
Ctrl-H  cursor left
Ctrl-I  cursor right
Ctrl-J  cursor down
Ctrl-K  cursor up
Ctrl-L  clear screen
Ctrl-M  carriage return
Ctrl-Q  cursor on
Ctrl-T  cursor off
Ctrl-^  cursor home
```

ESC	Teletext standard	Atmos / Telestrat	Phoenix
@	Alphanum black	Ink black	Alphanum black
A	Alphanum red	Ink red	red
B	Alphanum green	Ink green	green
C	Alphanum yellow	Ink yellow	yellow
D	Alphanum blue	Ink blue	blue
E	Alphanum magenta	Ink magenta	magenta
F	Alphanum cyan	Ink cyan	cyan
G	Alphanum white	Ink white	white
H	Flashing	Std/Single H/Steady	Flashing
I	Steady	Alt/Single H/Steady	Steady
J	End Box	Std/Dbl.Hght/Steady	---------- (not implemented)
K	Start Box	Alt/Dbl.Hght/Steady	---------- (not implemented)
L	Normal Size	Std/Single H/Flash	Normal Size
M	Double Height	Alt/Single H/Flash	Double Height
N	Double Width	Std/Dbl.Hght/Flash	---------- (not implemented)
O	Double Size	Alt/Dbl.Hght/Flash	---------- (not implemented)
P	Mosaic black	Paper black	Mosaic black
Q	Mosaic red	red	red
R	Mosaic green	green	green
S	Mosaic yellow	yellow	yellow
T	Mosaic blue	blue	blue
U	Mosaic magenta	magenta	magenta
V	Mosaic cyan	cyan	cyan
W	Mosaic white	white	white
X	Conceal Display	Text 60 Hz	Conceal Display

```
 Y | Contiguous Mosaics | Text  60 Hz        | Contiguous Mosaics
 Z | Separated Mosaics  | Text  50 Hz        | Separated Mosaics
 [ | ESC / Switch       | Text  50 Hz        | -------------  (not implemented)
 \ | Black Background   | Hires 60 Hz        | Black background
 ] | New Background     | Hires 60 Hz        | New Background
 ^ | Hold Mosaic char   | Hires 50 Hz        | Hold Mosaic char
 _ | Release Mosaic char| Hires 50 Hz        | Release Mosaic char
```

In mosaic mode, characters $40-$5F are still displayed as normal uppercase letters and symbols (“through characters”) while characters $20-$3F and $60-$7F give access to 64 mosaic characters (the separated set if ESC-Z has been selected, or the contiguous set).

80 columns mode: KayPro II compatibility

In 80 columns mode, unfortunately the EF9345-R005’s character set is not perfectly ASCII: curly brackets and tilde are for example replaced by vertical bars, which can be very annoying for a C programmer… it might then be a good idea to replace the Phoenix’ EF9345 chip by an older revision: the R003 version contains a perfectly standard ASCII set.

One can also note that all character codes in the range 128 to 255 are converted by the BIOS to the first 32 characters of the EF9345 chip (special punctuation characters). Unfortunately, this means that character codes from 128 to 255 are not stored in the EF9345’s video memory, and thus mosaic characters are not available in this 80-column mode.

```
Ctrl-H  backspace
Ctrl-I  tabulation
Ctrl-J  linefeed
Ctrl-K  line up
Ctrl-L  cursor right
Ctrl-M  carriage return
Ctrl-Q  cursor on
Ctrl-T  cursor off
Ctrl-W  clear end of screen
Ctrl-X  clear end of line
Ctrl-Z  clear screen
Ctrl-^  cursor home
```

```
ESC| Phoenix
---+-----------------------------------------------------
 = | cursor positionning (params: line, column)
 B | attribute off (0:colorset, 1:underline, 2:flashing, 3:negative)
 C | attribute on  (0:colorset, 1:underline, 2:flashing, 3:negative)
 E | line insert
 R | delete a line (and scroll the rest of the screen)
   |
 G | restore the saved position of the cursor
 H | change the color of the status line (param: @ABCDEFG) => colorset color
 L | change the color of chars on screen (param: @ABCDEFG)
 M | change the background color (and margins) (param: @ABCDEFG)
 S | save the current position of the cursor
```

The control characters are the same as the Kaypro II computer. The first series of escape sequences also start with the same command letter, but unfortunately the attributes are not exactly the same, nor numbered in the same way (the Kaypro uses 0: negative (inverse video), 1: reduced intensity, 2: blinking, 3: underlining, 4: cursor visible, 5: video mode, 6: remember/recover cursor position, 7: status line preservation on/off).

The missing intensity attribute has been a problem at first when using the Borland editors, but finally it can be nicely substituted by the colorset attribute : at boot time, the Phoenix will start in 80-column mode, with a cyan default color, and a white status line. This means that using the colorset attribute (“ESC C 0” sequence) will have the following characters displayed in white color.

A final remark: in 80-columns mode the attributes are handled as parallel attributes, so no blank character will be inserted when an attribute is changed.

G) Example of software configuration: Borland’s Turbo Pascal

Version 3.01 of Turbo Pascal for CP/M is a great IDE, with full-screen editor and a very fast compiler.

Starting the TINST.COM tool allows to configure the control sequences for the terminal, and alternative keys to control the editor.

H) First programs

First programs written (all with Turbo Pascal, sometimes with a few inline assembly routines):

- date program: allows to set the time and date of the RTC

- config program: allows to set all the BIOS configuration data, and save it to the battery-powered RTC’s RAM.

- Teletext library: set of Turbo Pascal procedures for sending the Teletext escape sequences. Also added a call to redefine the 96 characters patterns in 40-column mode.