PiDP-8/I

USER MANUAL
& SYSTEM DESIGN

DRAFT

(work in progress, v 20150816)

August, 2015

OBSOLESCENCE.WIX.COM/OBSOLESCENCE

CONTENTS

INTRODUCTION

The PiDP-8 is a modern remake of the Digital Equipment Corporation PDP-8/I. Design goal was to create a faithful but low-cost replica of the 1968 original. Moreover, the PiDP should replicate all stages in the PDP-8's development. Which is not trivial, because the PDP-8 series spanned a long period in computer history: from teletype & paper tape (1965) all the way through to hard disks and multi-user operating systems in 1979.

The PiDP-8 is based on the SimH PDP-8 emulator. SimH has been in development (in one form or another) since the 1960s, and is by far the most flexible and compatible PDP-8 emulation available. PDP-8 hardware setup is configurable from a basic 4K machine with paper tape reader, all the way up to a 32K multi-user system with 10MB RK05 disk cartridges. Also supported are DECtapes, floppy disks, fixed head disks, cassette and magnetic tape units, KL8JA additional terminals and various other hardware options such as the VC8E point-plot display.

Building on SimH's software flexibility, the PiDP-8 can be used either with serial terminals[1], operated stand-alone using the onboard HDMI/USB, or with remote terminal sessions on PCs over WiFi or Ethernet connections.

The original storage media used on PDP-8s has long since disappeared. For the PiDP, storage can be either tape/disk images stored on the internal SD card, or alternatively, USB sticks can be mounted as carriers for all original removable media – "USB paper tapes, USB disk cartridges, USB DECtapes".

This user manual is limited in scope: it **only** describes the operation of the PiDP-8.

- Of course, the underlying hardware is a Raspberry Pi, and the express design goal was to leave the PiDP usable as a normal Pi as well. Pi documentation can be found on raspberrypi.org.
- This manual provides a quick tour through the PDP-8 software universe, but the original DEC manuals are required to fully understand the PDP-8. Especially, the *Small Computer Handbook* and the other manuals listed on the Obsolescence Guaranteed website will be essential reading.
- Similarly, the SimH User Manual and SimH PDP-8 Manual will be useful to fully understand SimH's advanced features. They can be found on simh.trailing-edge.com.

[1] When the onboard serial port is enabled

USING THE PiDP-8

The PiDP-8 should feel very familiar to PDP-8/I users. Before describing some of the enhancements, it may be good to describe the differences first.

KNOWN INCOMPATIBILITIES

- On the front panel, 3-cycle and 1-cycle Data Breaks are not represented differently by the indicator leds, due to their implementation in SimH.
- For the same reason, single-stepping the subcycles *within* a single instruction (Sing Step) is not implemented. PDP-8 models after the 8/I also lack this feature, so it was felt that omitting it is no cardinal sin. Of course, single-stepping instructions (the Sing Inst switch) is implemented.
- For cost reasons, all front panel switches are on/off toggle switches. On the original 8/I, six of the switches were momentary toggles. On the PiDP-8, the user should toggle these 'on-off' manually or make his own modification[2]. The PiDP software converts these five switch signals to momentary signals automatically.

The PiDP (rather, the SimH engine) passes all relevant DEC diagnostics tests. See the SimH documentation for further details on compatibility.

BOOT PROCESS

The PiDP takes about 30 seconds to boot before the front panel lights come up. By default, OS/8 is booted up, but other configurations are switch-selectable. For instance, one configurations has only the RIM Loader present in memory. These settings can easily be modified as explained in later sections.

If the STOP switch is enabled, the PiDP will not start a PDP-8 session, but behave like a normal Raspberry Pi. Even then, a PDP-8 session can be initiated at any time by:

1. "sudo /opt/pidp8/etc/rc.pidp8 start"
2. or, alternatively, just run "sudo /opt/pidp8/bin/pidp8 <*config file*>"

Using the first option makes the PDP-8 run persistently, in parallel with the normal Raspberry Pi system. The PDP-8 keeps running even if a user logs out, and its terminal can be taken over by another physical terminal or from another user session.

[2] Unless springs are added into switch caps as a user modification

ADDITIONAL FRONT PANEL FUNCTIONS

The PiDP front panel provides some additional control functions compared to the original 8/I. All these functions are accessed through the Sing_Step switch.

(RE)BOOTING INTO DIFFERENT CONFIGURATIONS

The IF switches can be set to form an octal number. If a number between 1 and 7 is set **and** the Sing_Step switch is then toggled, the PiDP will read the correspondingly numbered boot script from the */opt/pidp8/bootscripts/* directory and reboot. By default, the following boot scripts are available when Sing_Step is toggled:

Octal	IF switches	Description
0	000	OS/8 on RK05 10MB disk cartridge
1	001	RIM Loader at 7756
2	010	TSS/8 multi-user system
3	011	OS/8 on DECtape
4	100	Spacewar! With vc8e output on localhost:2222
5	101	(empty)
6	110	(empty)
7	111	(empty)

When freshly booting up the PiDP, the above switch settings will also boot the PiDP into the stated configuration.

MOUNTING STORAGE MEDIA FROM USB STICKS

USB sticks are used as substitutes for all PDP-8 removable storage media, and a USB Hub can be thought of as the “Universal PiDP Storage Peripheral”. Mounting works as follows:

1. Insert the USB stick with the relevant image file (.pt, .dt, etc) on it
2. Select the device you want to mount it on by setting the Data Field switches:

DF Switches	Device Selected to mount on	File type
001	USB paper tape to High Speed Reader	.pt
010	USB paper tape to paper tape punch	.pt
011	Dectape on DT0 (TU55)	.dt
100	Dectape on DT1 (TU55)	.dt
101	8” floppy disk on RX0 (RX01/02)	.rx
110	8” floppy disk on RX1 (RX01/02)	.rx
111	Removable disk cartridge on RL0 (RL8a)	.rl

3. Toggle Sing_Step.

The PiDP will now scan all USB sticks and mount the first unmounted image file it finds for the selected device. Scanning requires the image file to have the extension as per the table above.

Notes:

- Multiple image files can reside on one USB stick, as long as they do not have the same extension. You can store any other files on the stick too, the PiDP will ignore them.
- The whole "PiDP Universal USB Storage Peripheral" concept can of course be ignored altogether: just use the SimH *attach <dev> <filename>* command to mount any image file.
- If you wish for another set of selectable devices through the DF field switches, the software is easily modified to accommodate for that.

SYSTEM SHUTDOWN

Set the Sing_Inst and Start switches, then toggle Sing_Step to shut down the PiDP. After 15 seconds, power can be cut off safely.

USING PiDP-8 AND LINUX CONCURRENTLY WITH 'SCREEN'

Effectively, the PiDP is both a PDP-8 and a normal Pi running concurrently. A freshly booted PiDP-8 will present the normal user login[3]. Straight after logging in, *screen* is called (it's added to .profile) to attach the user to the PDP-8. For a full overview of *screen* capabilities, read its documentation or google 'linux screen command examples' for a quick summary.

[3] Unless you modified it: user pi, password raspberry.

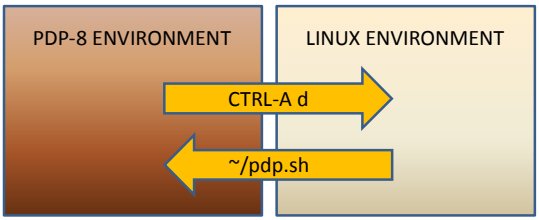
In essence, though, the simplest way of getting around is:

→ From within the PDP-8 environment, hit `Ctrl-A`, then `d` to detach yourself from the PDP-8 and enter into Linux.

→ From within Linux, enter `~/pdp.sh` to reattach yourself to the PDP-8. You can connect to the PDP-8 from where you want (command line, X Windows, remote ssh terminal session). If the PDP-8 screen is still attached to another user or terminal, you can also use `sudo screen -dr` to brute-force take over the PDP-8.

FUNCTION OVERVIEW

Front panel special functions: set function, then toggle Sing_Step switch



Switching between PDP-8 and Linux terminals

PREPARING TO USE THE PiDP-8 FOR THE FIRST TIME

Assembled & Tested PiDP-8's have their serial port enabled, and come with a USB-to-Serial adapter cable to allow a PC to both power the PiDP-8 and act as a serial terminal. No other cables are necessary. For Windows users:

- Download the puTTY terminal emulator from http://www.putty.org/.
- Install the PL2303HX USB driver from http://prolificusa.com/portfolio/pl2303hx-rev-d-usb-to-serial-bridge-controller/ .
- Check (using Windows' device manager) which COM port is assigned to the cable, and configure puTTY for a serial terminal connection at 115200bps.

Insert the USB cable into your PC. The PiDP-8 will power up and boot information will scroll by. Login is as user pi, password raspberry (as the boot image is an almost stock Raspberry Pi Raspbian distribution, you should change the password yourself).

ALTERNATIVE POWER AND TERMINAL OPTIONS

For PiDP-8 kit builders, the serial port modification[4] is very much optional. In fact, most kit builders are expected not to bother with it. In that case:

Power options:, either power the PiDP-8 from a micro USB cable[5] to the Raspberry Pi's power connector, or alternatively, use the GND and 5V pins from the serial port on the PiDP-8 PCB.

Terminal options: the PiDP can take ssh sessions from WiFi or Ethernet.

- Via Ethernet cable: A standard cable suffices (cross-over cable not necessary). See http://windows.microsoft.com/en-us/windows/using-internet-connection-sharing#1TC=windows-7 to configure your PC to share its network connection (assumed to be over WiFi) through to its Ethernet port, so the PiDP-8 can obtain an IP address and access the outside world.
- Via WiFi: Assembled & Tested PiDP-8s come configured to connect with WPA2 to SSID 'nerd7', password 'topsecret' if you are in a hurry. Configure your own

[4] See the Building Instructions on the web site for details on enabling the serial port.
[5] Depending on the height of the micro-USB connector, you may have to cut away the top of the connector's cable sleeve to make it fit into the PiDP-8 case. The cable has to make a sharp 90 degree turn to fit inside, pressing against the top of the case.

WiFi settings through the GUI (right-click WiFi icon at top right), or using the command line, as explained here: http://blog.self.li/post/63281257339/raspberry-pi-part-1-basic-setup-without-cables .

Either way, when the above is done, you need to find the IP address assigned to the PiDP-8. The primitive but fail-safe method is to use the Windows command line 'arp –a' to discover it, and then use puTTY to connect over ssh to that IP address. Note that you can also install VNC to work with the Raspberry Pi's GUI – no need to connect a HMDI monitor.

INSTALLING PiDP-8 SOFTWARE ON STANDARD RASPBIAN

For kit builders, or to upgrade to new issues of Raspbian, this section describes the process of installing PiDP-8 on a new Raspberry Pi. This example installs the software version used *without* the serial port. See the end of this section for the version required for serial port-enabled PiDP-8s. Both versions can be downloaded from the Obsolescence Guaranteed web site.

1-- **Untar pidp8.tgz (assumed to be on a USB stick in /media/usb0 in this example) into /opt** by: `cd /opt` ; `sudo tar -xvf /media/usb0/pidp8.tgz`
The result should be a /opt/pidp8 directory with source code and install scripts.

2-- **Compile the pidp8 software**: `cd pidp8/src` , then `sudo make`. This generates /opt/pidp8/bin/pidp8, the actual PiDP-8 program. Compile the scanswitch utility, which reads the frontpanel switches under Linux: `cd scanswitch`, then `sudo make`. This generates /opt/pidp8/bin/scanswitch.

3-- **Run the install script** `sudo /opt/pidp8/install/pidp8-setup.sh` to embed PiDP into the Raspberry Pi boot process. *You will need to have Internet access enabled before doing this*. Pidp8 will start at boot time as a detached process under the *screen* utility, running independently from any user logging in or out.
The install script installs Raspbian packages usbmount (to auto-mount USB sticks) and screen. **Crucially**, it disables the Pi's serial port so it will not interfere with the PiDP's gpio use[6]. Lastly, it creates a symlink in /etc/init.d to insert pidp8 into the boot process. You can delete the /etc/init.d/pidp8 symlink to undo this step if you wish.
After rebooting, the PiDP will automatically light up its front panel unless the STOP switch is enabled during bootup.

If you have modified your PiDP (and Pi) to use the serial port, download serial-pidp8.tgz instead of pidp8.tgz and follow exactly the same steps as described above, with the only difference that the install script is now called serial-pidp8-setup.sh. In fact, the only differences are:
- source code modification: #SERIALSETUP is defined in gpio.c .
- install script modification: script now obviously does not disable the gpio serial port.

[6] See the end of this chapter if you have the serial port modification done to your PiDP.

CUSTOMISING THE PiDP-8

→ In /opt/pidp8/bootscripts, you will see SimH script files numbered 1 to 7. Just change them to what you want the corresponding configuration (loaded with <IF switches> and <Sing_Step>) to be. The SimH manuals describe all configuration settings.

→ On the Raspberry Pi side of things, the default choice has been a wholly normal Raspbian distribution. Using the screen utility, you effectively have a PDP-8 and a Raspbian session running concurrently and can flip between the two.
But the standard Raspbian distribution is bloated and takes 30 seconds to boot. If all you want from the PiDP is a PDP-8, then it makes sense to use a 'light' Linux distribution such as Arch Linux, which can boot up in only a few seconds, and set it up to autoboot into the pidp8 program immediately without requiring a Linux login.

→ If you want to enable the serial port, please refer to the web site for the required modifications on your Raspberry Pi (remove two pull-up resistors) and on the PiDP-8 front panel PCB (use the C_COL jumper block, and add 2 resistors).

PiDP-8/I

USER MANUAL PART II:
SYSTEM DESIGN

DRAFT

(work in progress, v20150816)

August, 2015

OBSOLESCENCE.WIX.COM/OBSOLESCENCE

NOTE:

THIS DOCUMENT IS TO BE READ AS AN **APPENDIX TO THE NORMAL USER MANUAL**,

AVAILABLE FOR DOWNLOAD FROM THE WEB SITE
http://obsolescence.wix.com/obsolescence#!pidp-8-details/c1dem

SYSTEM DESIGN – OVERVIEW

From a **hardware** perspective, the PiDP is just a frontpanel add-on for a Raspberry PI. In the hardware section, the technical details of the front panel are explained. In fact, the front panel could just as easily be driven by any microcontroller, it only lights the leds and scans the switch positions.

From a **software** perspective, the PiDP is just a Raspberry Pi, running a modified version of the SimH emulator. SimH is modified to:

- drive the front panel lights in the appropriate manner - meaning it has instructions added to reflect the state of the PDP-8 CPU registers through the leds.
- let the CPU respond to the front panel switch settings.
- Add some special functions:
    - Mounting of disk and (paper) tape images stored on USB sticks;
    - (Re)boot the PDP-8 in different configurations;
    - Allow a power-down through the front panel.

The software design uses separate threads to run the PDP-8 emulation itself, and to maintain the physical front panel.

On the Obsolescence web site, you'll find the download section with all software, hardware (Kicad) and artwork (Inkscape) to replicate the PiDP-8:

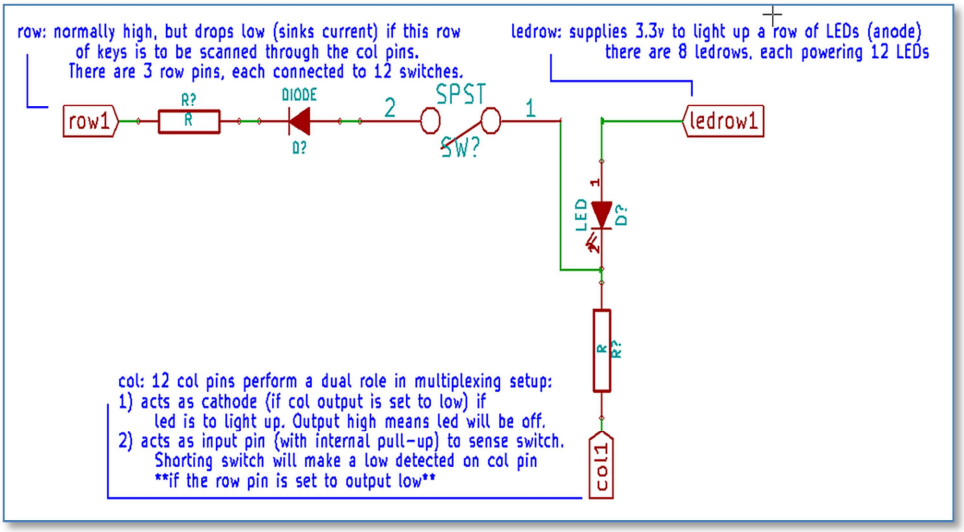
http://obsolescence.wix.com/obsolescence#!pidp-8-details/c1dem.

LICENSES AND PERMISSIONS

The PiDP-8 uses a modified version of the SimH emulator, and its use is thus bound by the Open Source license of SimH which can be found here: http://simh.trailing-edge.com/pdf/simh_doc.pdf. Written permission for using SimH within the PiDP-8 has been obtained.
The PiDP-8 uses original PDP-8 system and application software as per the 'Acknowledgements' section of the above document. This means the PiDP-8 can be used with PDP-8 software for hobbyist – i.e., non-commercial – purposes.

HARDWARE DESIGN

The Raspberry Pi (in its Plus and 2 versions) has a 40-pin GPIO connector that has just enough I/O pins to drive the front panel. The schematic used for the PiDP is actually taken from the venerable KIM-1 single board computer, and was used for the Obsolescence KIM Uno (a KIM-1 replica) too.
A multiplexing scheme is used to quickly light up alternating rows of leds in sequence. Doing so approximately 60 times per second, with the switch positions scanned in-between, the human eye sees the whole front panel light up.
Because the GPIO I/O pins are closely tied into the Pi's CPU chip, it only takes a few instructions to drive the front panel. The front panel is driven by a parallel process that reads the CPU status from SimH whilst it is running independently, and reflects the register states onto the front panel leds. The figure below shows the basic concept.



Three groups of GPIO pins are used:

- **8 'ledRow' pins**, each of which provides a collective power line ('+') to a row of 12 LEDs.
- **12 'column' pins**, which are connected to the cathodes ('-') of the individual 12 LEDs across all ledRows. So a row of LEDs is powered up when a ledRow pin is set to Output High. Which of the 12 LEDs actually lights up depends on whether its column pin is set to Output Low (LED off) or Output High (LED On).

- **3 'row' pins** each provide a power sink (0V) to a row of 12 switches. The column pins above, when flipped to Input mode with pullup resistors, can then sense which of the switches is 'on' (because that causes a short from row to its column pin overriding its weak pullup resistor).

The software quickly cycles through the 8 ledRows, setting each them High in sequence. Do it fast enough and the human eye sees all rows light up consistently without flickering. Then, the 12 column pins are quickly flipped over to being input pins to sense their attached switches, and the cycle starts afresh.

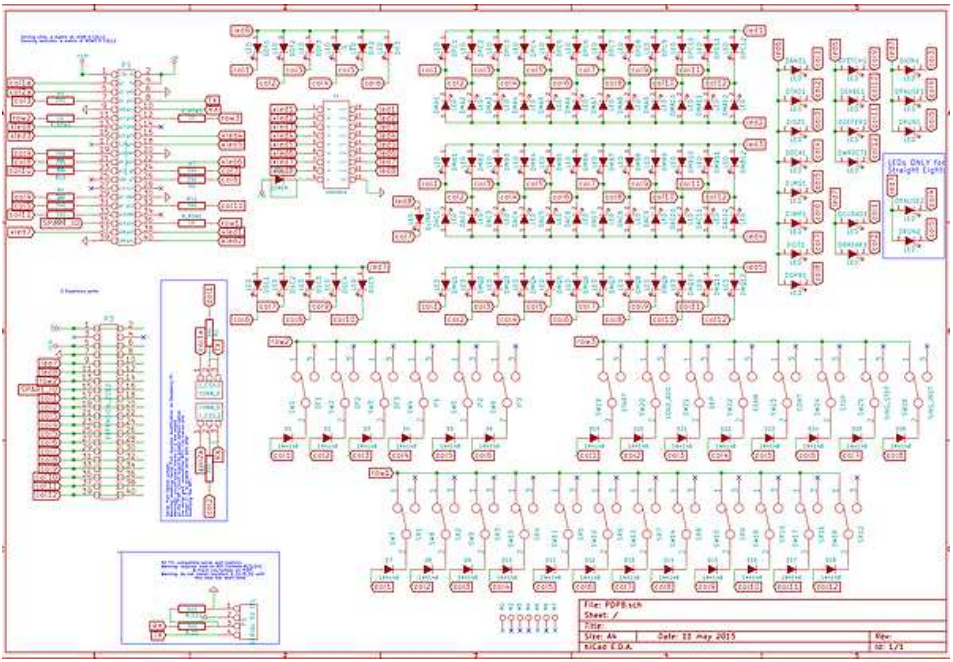
PROTECTING THE PI

Through diodes and resistors, the schematic is protected against programming errors, where output pins set high and low could short each other out. So all currents stay well below the Pi's specifications whatever a programmer does.

Another point: the GPIO of the Raspberry Pi cannot deliver much current. As long as each led's current is limited by a 1K resistor, all is well. But that only delivers 1mA of current through the leds. Bright enough for a front panel, but just barely so. To let the leds burn brighter, all the ledRow pins on the GPIO connector are buffered through a UDN2981A driver chip. That way, resistors of around 390K can be used to light up the leds much brighter. The UDN2981A has 8 inputs, connected to the GPIO, and switches its 8 outputs based on the signal it receives from the 8 GPIO pins. This driver chip can deliver much more current, and is fed on the 5V power line from the GPIO connector (note that everything else on the Pi and on the front panel is 3.3V). The voltage drop from this chip, and from the leds behind it means the GPIO pins on the other end of the schematic do not get exposed to the dangerous 5V levels - in fact, no more than 1.7V or so reaches them.

OVERALL SCHEMATIC

The picture below shows the full schematic, which is not much more than the above schematic wired through for 8 ledrows, 12 columns, and 3 (switch)rows. Two more things deserve mentioning: first, the UDN2981A can be left out completely, but then on the PCB, its pins 1-8 must be wired through to its pins 18-11. In that case the GPIO pins directly power the leds. Which is fine if you use 12 1K ohm resistors.



A full-resolution schematic, and the resulting board layout, can be found in the Kicad design files from the download section of the Obsolescence web site.

MODIFICATION: ENABLING THE SERIAL PORT OPTION

The designers of the Pi decided to add 1.8K pull-up resistors to I/O pins 2 and 3 (Broadcom numbering, not GPIO pin numbering). That means these two pins cannot be used straight away for the PiDP's purposes.

But, the front panel needs pretty much all the I/O pins on the Pi. If 2 and 3 cannot be used, 14 and 15 must be used. Why does this matter? Because I/O pins 14 and 15 can either be used as I/O pins, or act as a serial port (TX and RX). It is - obviously - nice to use this serial port for the PiDP. It adds one more way to hook up terminals, next to WiFi and Ethernet.

So, the user has to choose one of two options. The default is to use an unmodified Raspberry Pi, and give up the serial port. Alternatively, remove the pull-up resistors on the Pi and have the serial port available.

Users must choose the right jumper setting:

1. J1 and J2 jumpered to connect pins 1-2

   --> I/O 2 and 3 are not used as column pins. I/O 14 and 15 are used instead.

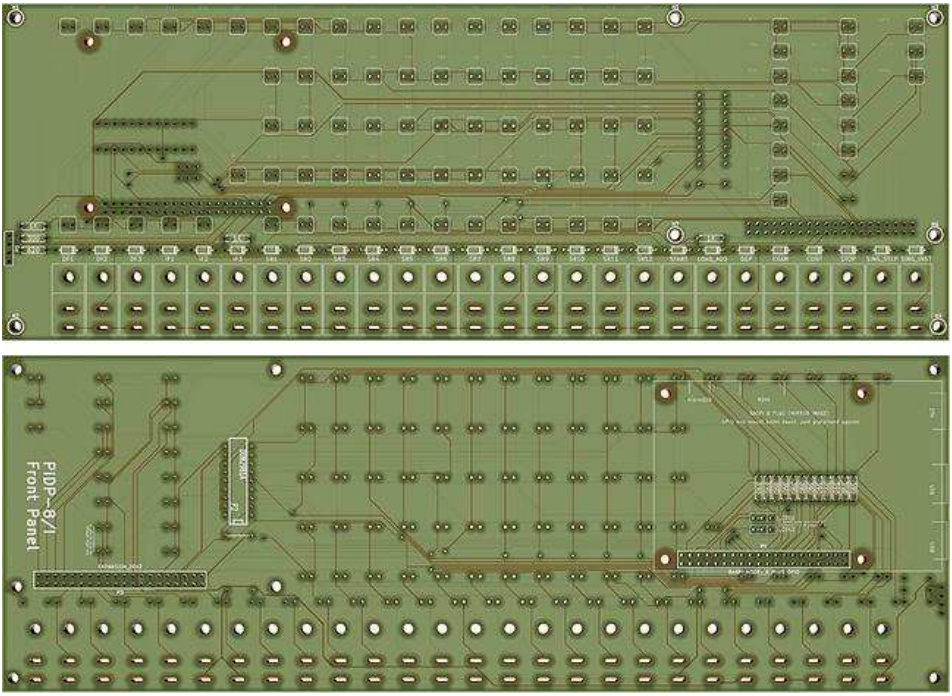
2. J1 and J2 jumpered to connect pins 2-3

   --> I/O 2 and 3 are used as column pins. I/O 14 and 15 can be used for serial port.

   --> the 1.8K pull-up resistors R23 and R24 must be removed from the Raspberry Pi.

PCB LAYOUT

The placing of LEDs and switches on the PCB is an exact 2:3 replica of the original PDP-8/I front panel. The pictures below whos the PCB front and back. For high resolution images, please refer to the Kicad design files in the download section of the web site.



COMPONENTS LIST

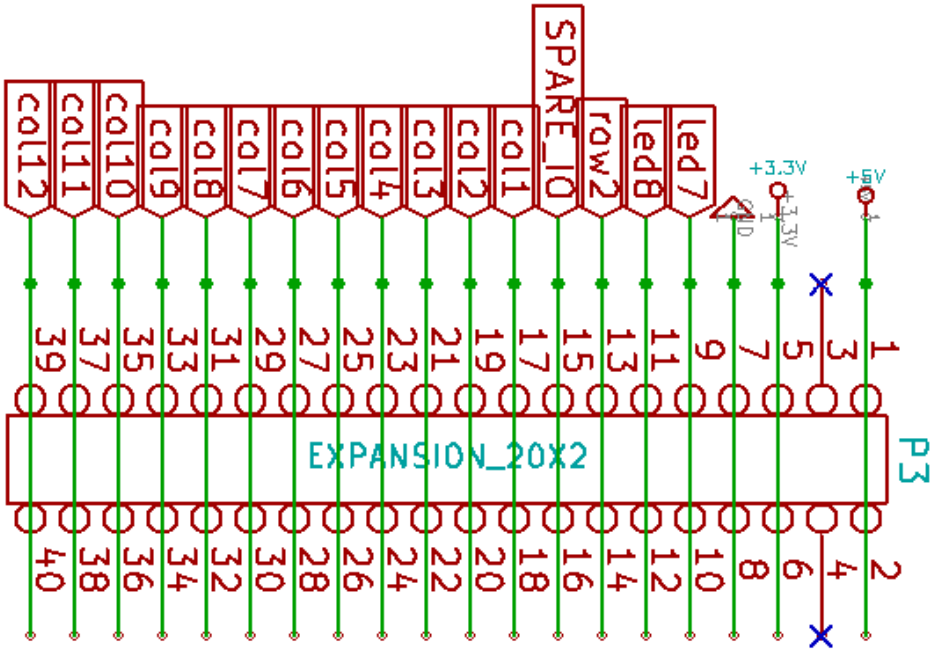
1× Raspberry Pi (Plus or 2) - needs 40 pin GPIO connector
26× Toggle switches
15× 1.2K resistors
26× 4148 Diodes
1× UDN2981A (optional)
1× PCB - Designed in Kicad/Open source hardware design
89× LEDs (5mm, High Brightness)

EXPANSION CONNECTOR(S)

A 2*20 pin header provides access to those signal lines that may be useful for peripheral devices. In fact, it serves as two identical single-row expansion connectors.

The 12 column pins are brought out, as well as the 2 row pins (for switches/inputs) and 1 ledrow pin (for LEDs/outputs) which are not fully occupied by switches/leds on the PiDP. For instance, row2 only has six PiDP switches attached to it. Six external switches, or inputs, can still be added. Take care with the ledrow pin. It comes straight out of the UDN2981A driver and has about 4 volts on it. Use a diode, maybe, to drop the voltage if a 3.3V device is attached to it.

Also brought out are the 5V and 3.3V power lines and of course GND. Most important, though, is the one single gpio pin that is left unused on the PiDP. This pin can be input, output, or both, under program control.



One obvious idea is to attach a spacewar controller/joystick to the expansion port. Look at gpio.c to understand how you can reach the in/output signals from software.

SOFTWARE DESIGN

CONCEPT

The software setup is as follows: when SimH starts up, it starts a second 'multiplexing' process which runs in parallel to SimH and which is set to be a high-priority 'Real Time' process. The main SimH program thread starts emulating the PDP-8, and that means there are a couple of variables containing the register values inside the PDP-8.

The multiplexing process, in gpio.c, does nothing more than continuously cycle through reading these register values, lighting up the 8 rows of LEDs accordingly, and sensing the 3 rows of switches.
The main SimH program meanwhile continues to emulate the PDP-8. This happens in pdp8_cpu.c, and the main function in there basically goes through the sequential logical operations as they are done in real PDP-8 hardware too.

The above description results in a SimH version that happily blinks the lights on the front panel without any modification to the SimH code itself. The multiplexing process just listens in and blinks its leds according to internal PDP-8 register values.

Some code has to be added in the main emulation function to read the front panel switch settings, and interrupt the instruction execution sequence accordingly. I.e., stop execution when the STOP switch is toggled.

MODIFICATIONS TO THE SIMH EMULATOR

In the startup part of SimH, scp.c, you'll see the multiplex process being started, and being closed down when the user exits SimH.
In the actual **emulator core, pdp8_cpu.c**, you'll see code added to make the PDP-8 registers visible to the multiplexing process, and code added to handle the switches on the front panel.
At the end of the pdp8_cpu.c, you'll see code that **mounts image files from USB sticks**. What is done here is nothing more than scan for image files on USB sticks, and mount them by sending an 'attach <dev> <filename> instruction to SimH, just like what a user could do manually on the command line.

Lastly, gpio.c is added to the SimH source code base. It contains the multiplexing process, with some initialisation of the I/O ports done before it enters its main loop.

The above should give you the background to understand the code modifications to SimH. Note that all modifications to SimH are added lines, clearly demarcated by lines to show where the added code starts and ends.

CHANGES TO THE STANDARD RASPBIAN DISTRIBUTION

See 'PiDP setup.doc' in the source code. Not much is done to the standard distribution: you need to install mountusb so USB sticks get mounted automatically. WiringPi is recommended for debugging because its gpio utility is helpful. But WiringPi's libraries are not used, they are too slow for the PiDP's purpose. All I/O access is done directly by writing GPIO registers.

PERSONAL OBSERVATIONS ON RASPBERRY PI GPIO

I almost discarded the Raspberry Pi as a basis for the PiDP because almost anytime the topic of real-time multiplexing on the Pi is mentioned, the consensus is that the Pi is too slow/Unix is unfit for running led multiplexing.

Those comments turned out to be nonsense. Linux on a Pi is more than fast enough to run a process that flips pin settings every 30ms with good consistency. In fact, that takes so little of the Pi's CPU power that you can not only run a PDP emulator concurrently, but also just use the Pi's GUI and run any other program you want. Even surf the web whilst the PDP emulator is running and the front panel blinks away.

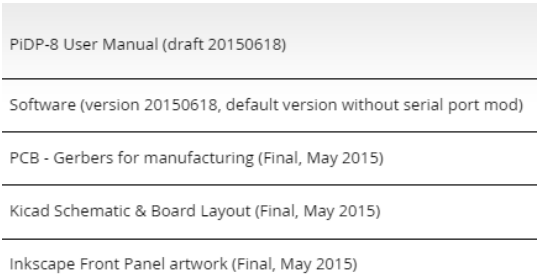
Another point is the idea that the GPIO port cannot provide enough mA to drive more than a few leds. But modern leds already light up on small amounts of current (1 mA is often enough), and as the PiDP (if you run it without the UDN2981A) proves: the GPIO ports has no problem driving **89** leds bright enough to use. Still, adding the UDN2981A allows the leds to be brighter, and that's nice enough to add that IC to the schematic.
Lesson learned: there's a lot of rubbish sold as received wisdom on the Raspberry Pi fora. I guess I should not have needed that lesson. With hindsight. But I wonder how many good projects never saw the light of day because of the false 'experts' on Raspberry fora!

APPENDIX: DOWNLOADS

The download link below contain the software and schematic/PCB/artwork files. The PiDP-8 is Open Source Hardware, and the downloads contain the complete information to create a working system:

http://obsolescence.wix.com/obsolescence#!pidp-8-details/c1dem

Available from the site are the following file archives:



Status as of August 2015:

- Software - beta version, check for updates now and then.
- Hardware and graphics - Final 1.0 versions

Comment on software: the software itself (pidp8) is pretty much final. However, the install script that goes with it is very much a temporary version. I messed around with someone else's good script.