PDP8-ES, a ESP32 **PDP8** simulator

1 Introduction.

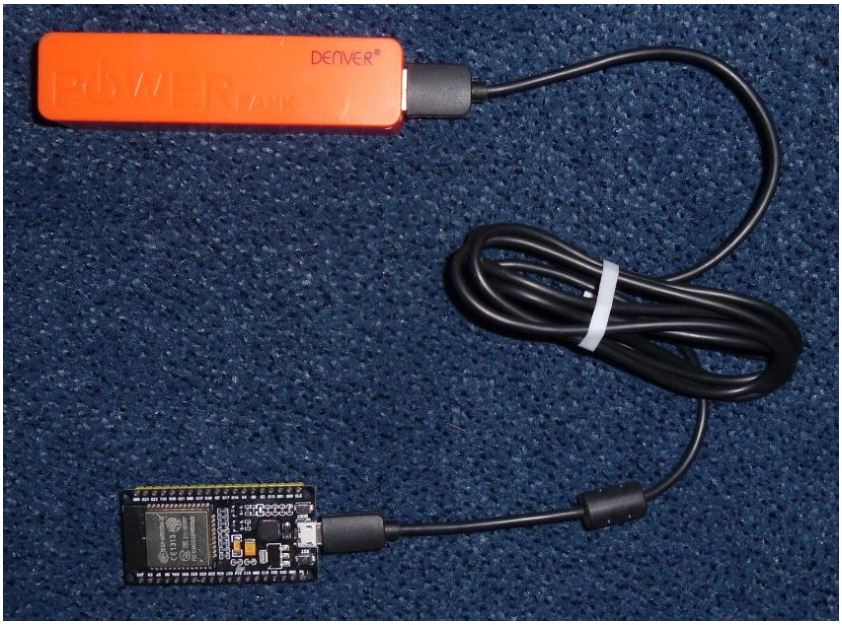
This document describes a pdp8 simulator that runs on an ESP32 SOC. The emulator simulates the pdp8 and some peripherals like an RK8E disk and a TU56 dectape drive capable of running the OS/8 operating system. The pdp8 is a well known minicomputer from the sixties. OS/8 is developed in the seventies.
On the internet there is plenty of information available about this remarkable machine.
The software is developed using the Espressif esp-idf toolchain. The speed of the simulated pdp8 is about as fast as an original pdp8/e machine, but disk/tape I/O is much faster. The simulator will emulate a pdp8 like this:

Four RK8E drives are simulated.

2 Hardware.

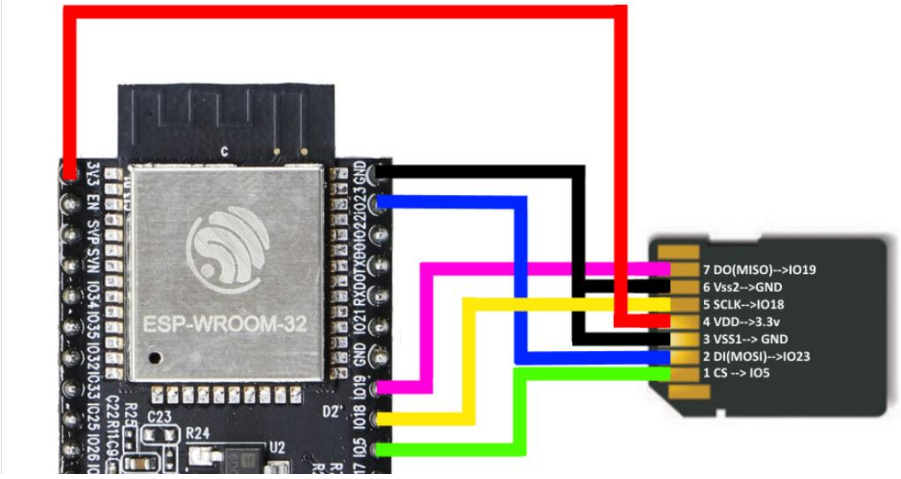
2.1 Minimal version.

The original hardware used for the simulator is an ESP32 development board with 4 MB flash. An SD-card can be connected to the VSPI bus of the ESP32 (pins 18,19,23) for storage of several OS/8 compatible images. The CS pin used for the SD card can be defined with “make menuconfig” in the esp-idf. The simulator can be used without the SD card. The console terminal is connected through a TELNET connecting using the WiFi capabilities of the ESP32. A minimal configuration could look like this:



2.2 SD Card (optional).

This is how an SD card can be connected:



Without the SD card the simulator can download an image of the OS/8 operating system. OS/8 will be saved in the flash memory of the ESP32 and can be booted afterwards.

This version is not longer supported. It is replaced by an ESP32 with OLED and 16 MB flash. The old project will remain on github in the archive directory.

2.3 Version with OLED and SD Card.

This version results in a very small PDP8. I call it the PDP8-ES. "ES" stands for "Extra Small", "ESp32" and "Ed Smallenburg". For this PDP8-ES I used a development board from Aliexpress "16 Mt bytes Pro ESP32 OLED V2.0 TTGO". This module has a build-ind OLED screen that is used to display the famous console lights of the PDP8 in "real time". It also has 128 Mbit flash instead of the standard 64 Mbit. This extra space is used to simulate more peripheral devices.

The result is a very small PDP8 (see picture). A small demo video can be seen at http://smallenburg.nl/pdp8/PDP8-ES.MP4.
The module has a build-in interface for a LiPo battery, so you can run your PDP8-ES completely wireless (for a limited time).

3 Partitioning.

The OS/8 block devices like RKA0:, RKB0:, DTA0: are emulated as partitions in the ESP32 module with 16 MB (128 Mbit) flash the partition table is:

```
# Name,   Type, SubType, Offset,    Size,     Flags
nvs,      data, nvs,     0x0009000,  0x004000,
otadata,  data, ota,     0x000D000,  0x002000,
phy_init, data, phy,     0x000F000,  0x001000,
factory,  app,  factory, 0x0010000,  0x100000,
ota_0,    app,  ota_0,   0x0110000,  0x100000,
ota_1,    app,  ota_1,   0x0210000,  0x100000,
RKA0,     data, 0x80,    0x0310000,  0x140000,
RKB0,     data, 0x80,    0x0450000,  0x140000,
RKA1,     data, 0x80,    0x0590000,  0x140000,
RKB1,     data, 0x80,    0x06D0000,  0x140000,
RKA2,     data, 0x80,    0x0810000,  0x140000,
RKB2,     data, 0x80,    0x0950000,  0x140000,
DTA0,     data, 0x81,    0x0A90000,  0x04C000,
DTA1,     data, 0x81,    0x0ADC000,  0x04C000,
#Free                    0x0B28000
#End+1                   0x1000000
```

A sector (1000 hexadecimal bytes = 4kB) in the flash memory can hold 21 pdp8 pages of 128 packed words. A sector in flash can only be written to as a whole block of 4 kB. To minimize the number of writes (and reads), a cache of at least 16 sectors are buffered in the simulator. The dirty cache buffers (caused by write operations) will be saved in flash every 5 minutes or by using the "PO" command in the console.

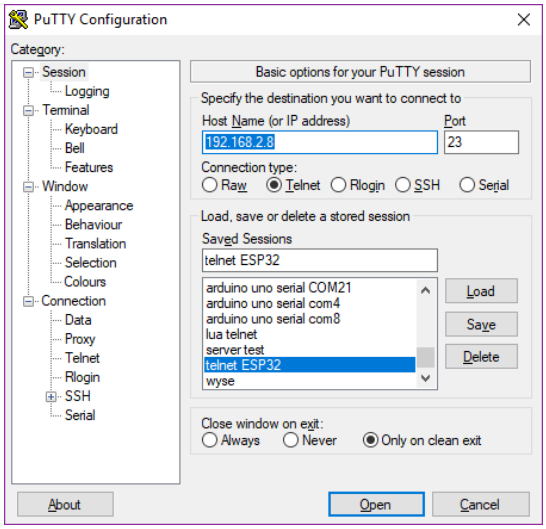
At least the RKA0 partition must be filled by a proper image in order to run OS/8 on the simulator. The image can be downloaded from the Internet through the build-in http client or loaded from an SD card.

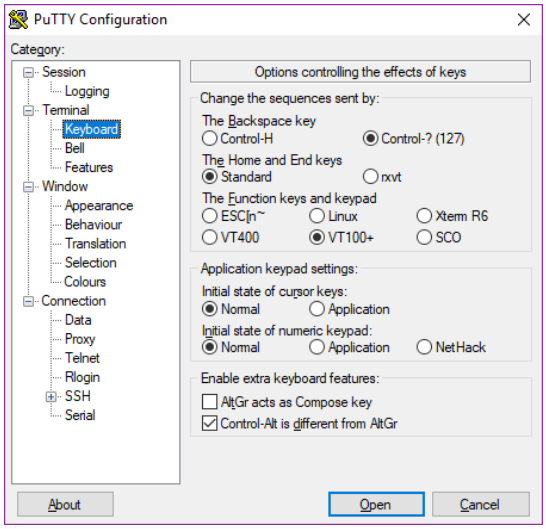
A fully patch RKA0/RKB0 image will be available at http://smallenburg.nl/pdp8/os8patched.rk05, ready to be transferred to an SD card.

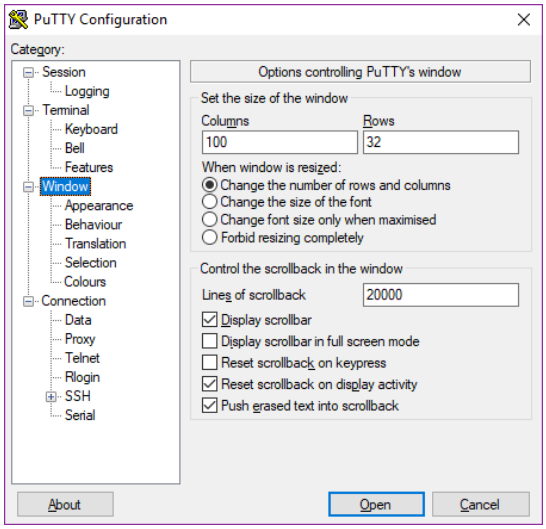
4 Managing the simulator.

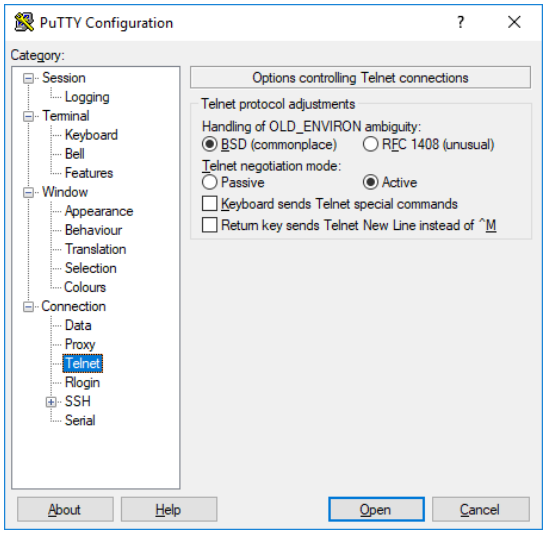
4.1 TELNET client.

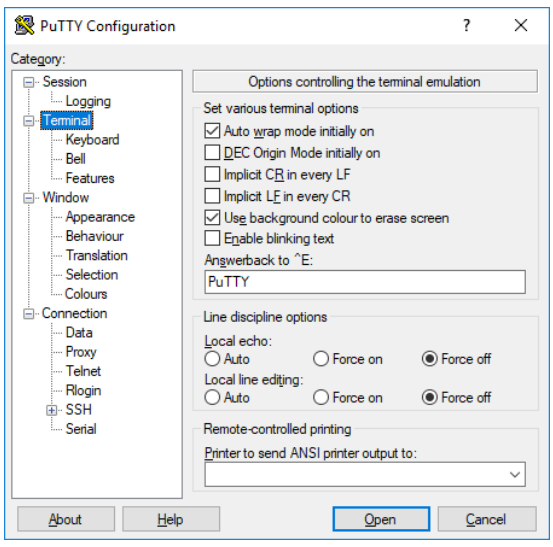
On (re)start of the ESP32, the simulator does not start automatically. The user must make a TELNET connection to the ESP32. A good TELNET client for this purpose is “PuTTY” (see http://putty.org).
The last digit of the IP address will be displayed in the MQ register of the console in octal form. I my case it was 10 (octal), so the address is 192.168.2.8.
Assuming an IP address of 192.168.2.8, the settings for a connection are:









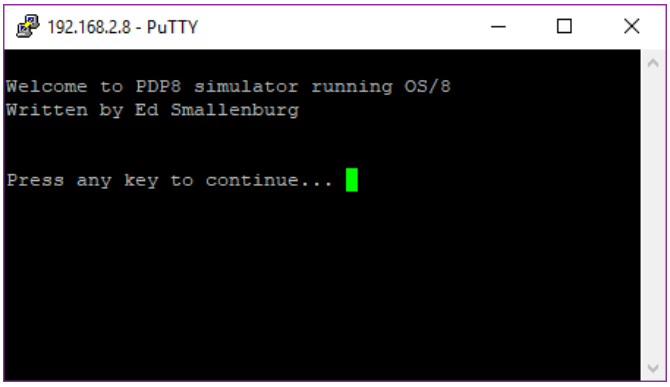


You may save the settings for convenience.

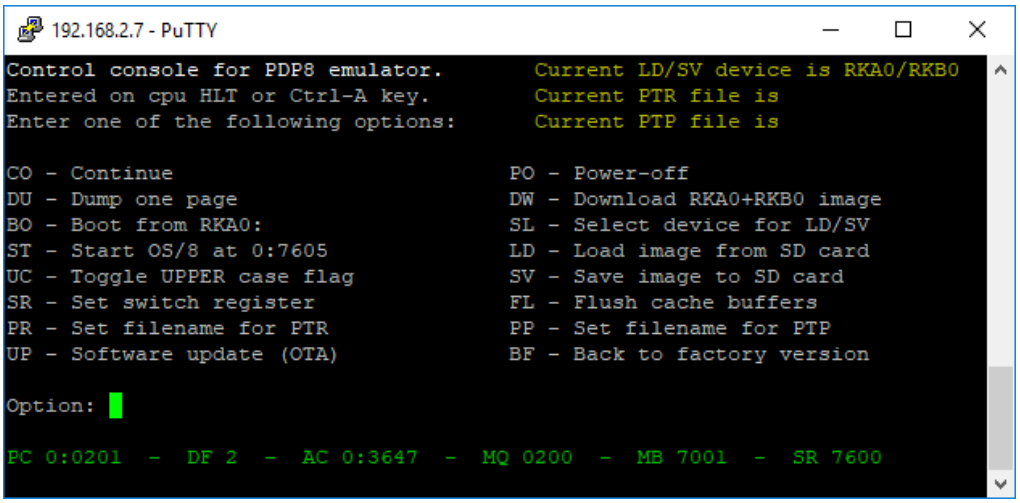
4.2 Control Console.

Connect to the simulator through the build-in Control Console.

Once connected, you will see a screen like this:



An extra line is added if an SD card is connected.
After pressing a key, the simulator console will be shown:



This screen can be activated any time by pressing Ctrl-A on the keyboard. A HLT instruction or an illegal IOT will also bring up this screen.
You may adjust the size of the console screen if filenames are too long to fit the screen.

The most obvious command is “BO”, but for the first time you have to configure your PDP8. Possible commands are listed below. Command can have one or two parameters.

- CO – This will continue the PDP8 after a Ctrl-A interrupt.
- DU – This will dump the contents of one PDP8 page on the screen. Parameters may be the data field and the start address of the dump. If no parameters are supplied, the next page will be dumped.
  Examples: “DU 3 200”, “DU 1000”, “DU”.
- BO – This will boot the RKA0: disk and start OS/8.
- ST – This will start OS/8 at 07605 after a Ctrl-A interrupt. Parameters may be field and address of a different start point.
  Examples: “ST”, “ST 0 7600”, “ST 0200”.
- UC – This will toggle the Uppercase flag. Some versions of OS/8 expect only uppercase characters. This function may be handy for this situation.

- SR – Will set the pdp8 switch register according the parameter. The "LAS" instruction will read this number in the accumulator.
  Example: "SR 7600"
- PR – Set the input file (on SD card) for the simulated paper tape reader. The current input file will be visible at the top of the screen.
- UP – This will update the software through OTA. By default this software is loaded from www.smallenburg.nl/pdp8/esp-pdp8.bin, but you may specify a different location as a parameter.
- PO – Power off. This will write cached data to the simulated drives and the ESP32 is set to sleep. Wake up by the RESET or BOOT button of the development module. Not that disconnecting the power without this command does not write the cache to the simulated disks/tapes, and may therefore cause file inconsistency.
- DW – This will download a more or less working OS/8 system from the internet at **www.pdp8online.com/ftp/images/os8/diag-games-kermit.rk05**. You may also supply a different URL as a parameter for this command. Other images are available if an SD card (with the right data on it) is connected.
- SL – This will select a pdp8 device for the LD and SV commands. If no parameter is supplied, a list of the available devices will be shown.
- LD – This will copy an image from the SD-card to the selected pdp8 device. For RKA0/RKB0, the image is usually the size of one or two disks. The reason is that the disk was divided into 2 "sides" as the whole disk was too big to be addressed as one device (more than 4096 blocks). Without a parameter, the console will show all the matching (see SL command) images on the SD card like this:
  os8.rk05
  diag-games-kermit.rk05
  To load an image, select the right input file according to the directory listing.
  Example: "LD os8.rk05". The extension of the file will be forced to the standard extension, like ".rk05" for a RKA0/RKB0 image.
  Note: a load from existing .rx01 files does not work yet. However, you may save an rx01 image and read it back using LD.
- SV – Save the image of the selected device to the SD card. This is the reverse of "LD". You have to specify a filename. The extension will be forced to a standard extension.
- FL – This will flush the cached data to the pdp8 devices. Never turn the power off without a flush! You may also use the "PO" command.
- PP – Set the output file (on SD card) for the simulated paper tape punch. Specify the filename in the parameter. The current output file is visible at the top of the screen.
- BF – Ignore the OTA updates and go back to the "factory" software. This step is necessary when you updated the software by "UP" and you want to upload a new version to the ESP32 by the "make flash" command.

5 Running OS/8.

5.1 Unpatched OS/8 images.

An unpatched RK8E OS/8 image will run on the simulator. But only RKA0 is available in this case. You can make RKB0 accessible by making a simple patch. This is the procedure:

1. Find out what the device number is for RKB0. Use RESORC.SV for this purpose:
   ```
   .RESORC /E
   ```
   This will show a table like:
   ```
   #  NAME TYPE MODE SIZ BLK KIND U V ENT USER
   01 SYS  RK8E RWF 3248 SYS      0 C  07
   02 DSK  RK8E RWF 3248 SYS      0 C  07
   03 RKA0 RK8E RWF 3248 16  RK05 0 A  20
   04 RKB0 RK8E RWF 3248 16  RK05 0 A  21
   05 RKA1 RK8E RWF 3248 16  RK05 1 A  22
   06 RKB1 RK8E RWF 3248 16  RK05 1 A  23
   07 RXA0 RX8E RWF  494 17         E  30
   ```
2. Look at the device number (column 1) for device RKB0. In this example it is 4.
3. Start ODT and open location 17646 + DEVNR, in this example 17652 and change its contents to 7613. Exit ODT afterwards by pressing Ctrl-C:
   ```
   . ODT
   17652/0000 7613
   .^C
   .
   ```
4. Now RKB0 is accessible.
5. Optional, you can make DTA0 accessible by putting the number 7617 into 17646+(devnr of DTA0).

5.2 Patched OS/8 images.

In the data directory is a fully patched OS/8 image available (os8patched.rk05).

5.3 Some handy commands:

- SET TTY WIDTH 80
  This prevents extra new lines to be printed if lines are longer than 72 characters.
- SET TTY NO PAUSE
  This will prevent pauses if text scrolls over the screen. Will not work with all OS/8 versions.
- DIR
  This will give you a directory listing. Also “DIR DTA0:” or “DIR RKB0:” Note that device names end with a semicolon to distinguish if there would be confused with a filename.
- HELP
  This will give you some help texts.

5.4 OS/8 programs incompatibility.

Some programs will not run on this simulator. Specially programs that uses I/O directly. Direct I/O for extended memory handling, papertape and teletype I/O are emulated, but if a program tries to execute IOTs for disk or dectape access directly, the simulator will stop with information about the failed instruction.
So for example DTCOPY.SV will not work. Programs that use the interrupt facility will also not work.
A PATCH.PA program is supplied in the tools directory that will patch CCL, DIRECT, RESORC and PIP for this simulator.

5.5 CCL.SV.

The OS/8 date was originally designed to run from 1-JAN-70 to 31-DEC-77. This limit was caused by having just 3 bits to store the year. Here the year offset was 1970. Later on, the limit of the year was extended to 1999 by using 2 additional bits at location 07777, bits 3 and 4. The simulator tries to set the date automatically on boot, at location 17666 and 07777. For 2017, this will result in something like Sunday August 23, 1987.
So we have to change the offset for the year from 1970 to 2000. I patched CCL.SV for this purpose. Now, since a long time, it prints the correct date:

```
.DATE
Wednesday August 23, 2017
```

Note that setting the date is no longer necessary and maybe impossible.
I also patched DIRECT.SV so it will print the right date:

```
.DIR

 17-Jan-18

K12DEC.SV   5            RXCOPY.SV   9 04-Jan-11  LOAD  .LS   8
K12MIT.SV  33            FLOP  .HN   3 04-Jan-11  TEST  .SV   2 12-Jan-18
K12DJG.SV  33            LCSYS .BI   3            PASCAL.BN  63 03-Jan-18
K12DEB.SV   4            UCSYS .BI   3            TEST  .PA   2 12-Jan-18
K12ENB.SV   4            BASIC .WS  11            TEST  .BN   1 12-Jan-18
BRTS  .SV  15 01-Dec-13  SET   .SV  20 04-Jan-11  FAKE  .PA   7 16-Jan-18
F4    .SV  20 21-Jul-15  RTFLOP.SV  15 04-Jan-11  FAKE  .BN   1 16-Jan-18
FRTS  .SV  26 04-Oct-17  WPFLOP.SV  14 04-Jan-11  PATCH .PA   5 16-Jan-18
PASS2 .SV  20 21-Jul-15  RESORC.SV  14 04-Jan-11  PATCH .BN   2 16-Jan-18
PASS2O.SV   5 21-Jul-15  DECX8 .SV  43 24-Mar-17  X     .DI  15 16-Jan-18

 171 Files in 2746 Blocks -  446 Free blocks
```

5.6 F4.SV / FRTS.SV.

The Fortran compiler seems to work. However I was not able to run the famous ADVENTURE program. Compiling the source code succeeded, but there was an error on running the program.

5.7 BASIC.SV

The RUN command in basic does not work. But you may run your basic programs by:

```
.COMPILE SPACWR.BA
```

5.8 Programs that run normally.

PAL8.SV	ABSLDR.SV	TECO.SV
EDIT.SV	FUTIL.SV	FORT.SV
SABR.SV	LOADER.SV	PIP.SV
CCL.SV	BITMAP.SV	BUILD.SV

6 Fake device handlers.

6.1 Block drivers.

The OS/8 block drivers for the simulated devices like RK8E, DTA0,…. are not emulated by their IOTs. Instead their functions (read and write) are simulated. The IOTs 6770, 6771, 6772, 6773, 6774 and 6775 are used for this purpose. A special device handler has been made that can be used in "BUILD". It forms a "fake" handler for 7 devices: SYS, RKA0, RKB0, DTA0, DTA1, RXA0 and TMP0. Below is the listing of the source code of this handler.

```
/ FAKE.PA
/
/ FAKE HANDLER FOR PDP8 SIMULATOR
/ ED SMALLENBURG, 16-JAN-2018
/
        VERSION="C&77
        *0

        DECIMAL;RKLEN=3248;OCTAL
        DECIMAL;DTALEN=737;OCTAL

        -11      /9 HANDLERS: SYS,RKA0,RKB0,RKA1,RKB1,RKA2,RKB2,DTA0,DTA1
        DEVICE FAKE;DEVICE SYS ;4231;2007;0;RKLEN
        DEVICE FAKE;DEVICE RKA0;4231;1007;0;RKLEN
        DEVICE FAKE;DEVICE RKB0;4231;RKB0H&177+1000;0;RKLEN
        DEVICE FAKE;DEVICE RKA1;4231;RKA1H&177+1000;0;RKLEN
        DEVICE FAKE;DEVICE RKB1;4231;RKB1H&177+1000;0;RKLEN
        DEVICE FAKE;DEVICE RKA2;4231;RKA2H&177+1000;0;RKLEN
        DEVICE FAKE;DEVICE RKB2;4231;RKB2H&177+1000;0;RKLEN
        DEVICE FAKE;DEVICE DTA0;4161;DTA0H&177+1000;0;DTALEN
        DEVICE FAKE;DEVICE DTA1;4161;DTA1H&177+1000;0;DTALEN
        BOOT-BLAST
        RELOC 0

BOOT,   TAD I BOOTX1
        DCA I BOOTX2
        TAD I BOOTX3
        CDF 10
        DCA I BOOTX4
        CDF 0
        TAD BOOTX2
        SZA CLA
        JMP BOOT
        JMP I B7605
BOOTX1, 177
BOOTX2, 7577
BOOTX3, 46
BOOTX4, 7646
        ZBLOCK 30-.
B7605,  7605
BLAST,  RELOC

          *200
          RELOC 7600

          ZBLOCK 7
SHNDLR, VERSION             /SYSTEM AND RKA0: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD SHNDLR        /POINTER TO PARAMETERS
          6770              /SIMULATES RKA0: READ/WRITE
/NO RETURN HERE, SIMULATOR WILL RETURN TO CALLERS RETURN ADDRESS

RKB0H,    VERSION           /RKB0: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD RKB0H /POINTER TO PARAMETERS
          6771              /SIMULATES RKB0: READ/WRITE
/NO RETURN HERE

RKA1H,    VERSION           /RKA1: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD RKA1H /POINTER TO PARAMETERS
          6772              /SIMULATES RKA1: READ/WRITE
/NO RETURN HERE

RKB1H,    VERSION           /RKB1: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD RKB1H /POINTER TO PARAMETERS
          6773              /SIMULATES RKB1: READ/WRITE
/NO RETURN HERE

RKA2H,    VERSION           /RKA2: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD RKA2H /POINTER TO PARAMETERS
          6774              /SIMULATES RKA2: READ/WRITE
/NO RETURN HERE
RKB2H,    VERSION           /RKB2: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD RKB2H /POINTER TO PARAMETERS
          6775              /SIMULATES RKB2: READ/WRITE
/NO RETURN HERE

DTA0H,    VERSION           /DTA0: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD DTA0H /POINTER TO PARAMETERS
          6776              /SIMULATES DTA0: READ/WRITE
/NO RETURN HERE

DTA1H,    VERSION           /DTA1: ENTRYPOINT
          CLA CLL           /GUARD AGAINST NON-ZERO AC
          TAD DTA1H /POINTER TO PARAMETERS
          6777              /SIMULATES DTA1: READ/WRITE
/NO RETURN HERE

          RELOC
          $
```

The configuration after “BUILD” looks like this:

```
.RESORC /E

171 FILES IN 2746 BLOCKS USING 5 SEGMENTS
446 FREE BLOCKS (5 EMPTIES)

#  NAME TYPE MODE SIZ BLK KIND U V ENT USER
01 SYS  RK8E RWF 3248 SYS      0 C  07
02 DSK  RK8E RWF 3248 SYS      0 C  07
03 TTY  TTY  RW       16+ KL8E   E 176
04 PTP  PTP  W        17  PT8E   A  00
05 PTR  PTR  R        17  PT8E   A 112
06 RKA0 RK8E RWF 3248 SYS      0 C  07
07 RKB0 RK8E RWF 3248 SYS      1 C  13
10 RKA1 RK8E RWF 3248 SYS      1 C  17
11 RKB1 RK8E RWF 3248 SYS      1 C  23
12 RKA2 RK8E RWF 3248 SYS      1 C  27
13 RKB2 RK8E RWF 3248 SYS      1 C  33
14 DTA0 TC08 RWF  737 SYS      1 C  37
15 DTA1 TC08 RWF  737 SYS      1 C  43

FREE DEVICE SLOTS: 02,  FREE BLOCK SLOTS: 06
OS/8 V3T
.
```

6.2 Fake PTR.

A paper tape reader (PTR) can be very handy to load source code to a pdp8 file. Also binary files (with the “.bn” extension) can be read from paper tape.
The simulator has the possibility to connect a file on SD card to the virtual paper tape reader. The IOTs for the paper tape reader can read from this file. At end of file, a Ctrl-Z character will be the result of the RRB instruction.
The filename for the simulated PTR can be supplied in the PR command. If the filename is omitted, the directory of the SD card is presented.
In the simulator a file can be read, for example, like:

```
.R PIP
*XXX.PA<PTR:
```

6.3 Fake PTP.

The PTP: handler is simulated to redirect the output to a SD card file. The name for the file must be specified in the control console. This makes it possible to get source files (or binaries) out of the simulator.
Output to paper can be initiated like this:

```
.R PIP
*PTP:<XXX.PA
```

7 More information.

7.1 The best website for software and manuals is:

http://www.vandermark.ch/pdp8

7.2 pdp8 online home page.

http://www.pdp8online.com

8 Quick start.

The quickest way to get the system to work:

1. Copy the project to your esp-idf project space. Mine is at C:\msys32\home\Ed\esp.
   You should see something like:

```
Ed@SH4 MINGW32 ~/esp/esp_pdp8
$ ls -l
totaal 46
drwxr-xr-x 1 Ed Geen     0 17 jan 11:23 build
drwxr-xr-x 1 Ed Geen     0  5 jan 14:10 data
drwxr-xr-x 1 Ed Geen     0 17 jan 10:42 doc
drwxr-xr-x 1 Ed Geen     0 17 jan 11:29 main
-rw-r--r-- 1 Ed Geen   180  2 aug 16:33 Makefile
-rw-r--r-- 1 Ed Geen   892 16 jan 15:24 partitions.csv
-rw-r--r-- 1 Ed Geen   298  1 sep 09:57 README.md
-rw-r--r-- 1 Ed Geen 12934  8 jan 14:57 sdkconfig
drwxr-xr-x 1 Ed Geen     0 16 jan 10:51 tools

$
```

2. Run “make menuconfig” to define your preferences.
3. Run “make flash monitor” to compile, link and upload the software tot he ESP32. Compiling takes a long time fort he first time. After startup you should see the logging as shown in the next chapter.
4. Use a telnet session to connect tot he IP-address of the ESP32.
5. Run the “DW” command in the pdp8 control console to download a patched OS/8 to RKA0:/RKB0:.
6. Run the “BO” command to bootstrap the pdp8.
7. You are now in OS/8 mode. Type the command “DIR” to show the RKA0: directory.

9 Logging during start-up.

This will be logged on the serial output during start-up. This may vary according the debugging level set in the configuration.

```
ets Jun  8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:5564
load:0x40078000,len:0
load:0x40078000,len:13712
entry 0x40079020
I (29) boot: ESP-IDF v3.1-dev-78-g496f6d27 2nd stage bootloader
I (29) boot: compile time 16:24:11
I (29) boot: Enabling RNG early entropy source...
I (34) boot: SPI Speed      : 40MHz
I (39) boot: SPI Mode       : DIO
I (43) boot: SPI Flash Size : 16MB
I (47) boot: Partition Table:
I (50) boot: ## Label            Usage          Type ST Offset   Length
I (58) boot:  0 nvs              WiFi data        01 02 00009000 00004000
I (65) boot:  1 otadata          OTA data         01 00 0000d000 00002000
I (73) boot:  2 phy_init         RF data          01 01 0000f000 00001000
I (80) boot:  3 factory          factory app      00 000 00010000 00100000
I (87) boot:  4 ota_0            OTA app          00 10 00110000 00100000
I (95) boot:  5 ota_1            OTA app          00 11 00210000 00100000
I (102) boot:  6 RKA0             Unknown data     01 80 00310000 00140000
I (110) boot:  7 RKB0             Unknown data     01 80 00450000 00140000
I (118) boot:  8 RKA1             Unknown data     01 80 00590000 00140000
I (125) boot:  9 RKB1             Unknown data     01 80 006d0000 00140000
I (133) boot: 10 RKA2             Unknown data     01 80 00810000 00140000
I (140) boot: 11 RKB2             Unknown data     01 80 00950000 00140000
I (148) boot: 12 DTA0             Unknown data     01 811 00a90000 0004c000
I (155) boot: 13 DTA1             Unknown data     01 81 00adc000 0004c000
I (163) boot: End of partition table
I (167) boot: Defaulting to factory image
I (172) esp_image: segment 0: paddr=0x00010020 vaddr=0x3f400020 size=0x17df4 ( 97780) map
I (215) esp_image: segment 1: paddr=0x00027e1c vaddr=0x3ffb0000 size=0x03304 ( 13060) load
I (220) esp_image: segment 2: paddr=0x0002b128 vaddr=0x40080000 size=0x00400 (  1024) load
0x40080000: _iram_start at C:/msys32/home/Ed/esp/esp-idf/components/freertos/xtensa_vectors.S:1685

I (222) esp_image: segment 3: paddr=0x0002b530 vaddr=0x40080400 size=0x04ae0 ( 19168) load
I (238) esp_image: segment 4: paddr=0x00030018 vaddr=0x400d0018 size==0x69e58 (433752) map
0x400d0018: _stext at ??:?

I (390) esp_image: segment 5: paddr=0x00099e78 vaddr=0x40084ee0 size=0x0db10 ( 56080) load
0x40084ee0: spi_intr at C:/msys32/home/Ed/esp/esp-idf/components/driver/spi_master.c:477

I (413) esp_image: segment 6: paddr=0x000a7990 vaddr=0x400c0000 size=0x00064 (   100) load
I (424) boot: Loaded app from partition at offset 0x10000
I (424) boot: Disabling RNG early entropy source...
I (425) cpu_start: Pro cpu up.
I (429) cpu_start: Starting app cpu, entry point is 0x400810d4
0x400810d4: call_start_cpu1 at C:/msys32/home/Ed/esp/esp-idf/components/esp32/cpu_start.c:222

I (416) cpu_start: App cpu up.
I (439) heap_init: Initializing. RAM available for dynamic allocation:
I (446) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (452) heap_init: At 3FFCAE98 len 00015168 (84 KiB): DRAM
I (458) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (465) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (471) heap_init: At 400929F0 len 0000D610 (53 KiB): IRAM
I (477) cpu_start: Pro cpu start user code
I (160) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu__start: Starting scheduler on APP CPU.
I (230) [PDP8]: Booted from factory partition
I (240) gpio: GPIO[5]| InputEn: 0| OutputEn: 1| OpenDrain: 0| Pullup: 0| Pulldown: 0| Intr:0
I (330) wifi: wifi firmware version: f204566
I (330) wifi: config NVS flash: disabled
I (330) wifi: config nano formating: disabled
I (330) system_api: Base MAC address is not set, read default base MAC address from BLK0 of EFUSE
I (340) system_api: Base MAC address is not set, read default base MAC address from BLK0 of EFUSE
I (350) wifi: Init dynamic tx buffer num: 32
I (350) wifi: Init data frame dynamic rx buffer num: 32
I (360) wifi: Init management frame dynamic rx buffer num: 32
I (360) wifi: wifi driver task: 3ffd7630, prio:23,, stack:4096
I (370) wifi: Init static rx buffer num: 10
I (370) wifi: Init dynamic rx buffer num: 32
I (370) wifi: wifi power manager task: 0x3ffdc270 prio: 21 stack: 2560
I (380) [PDP8]: Setting WiFi configuration SSID NETGEAR-11...
I (430) phy: phy_version: 366.0, ba9923d, Oct 31 2017, 18:06:17, 1, 0
```

```
I (430) wifi: mode : sta (30:ae:a4:37:32:60)
I (1040) wifi: nn:5 0, o:1 0, ap:255 255, sta:5 0, prof:1
I (2020) wifi: state: init -> auth (b0)
I (2030) wifi: state: auth -> assoc (0)
I (2030) wifi: state: assoc -> run (10)
I (2050) wifi: connected with NETGEAR-11, channel 5
I (4790) event: sta ip: 192.168.2.7, mask: 2255.255.255.0, gw: 192.168.2.254
I (4790) [PDP8]: Connected to AP
I (4790) [PDP8]: 0 - RKA0 mounted, size 13A000 bytes (3297 OS/8 blocks) 3248 blocks used
I (4800) [PDP8]: 1 - RKB0 mounted, size 13A000 bytes (3297 OS/8 blocks) 3248 blocks used
I (4810) [PDP8]: 2 - RKA1 mounted, size 133A000 bytes (3297 OS/8 blocks) 3248 blocks used
I (4820) [PDP8]: 3 - RKB1 mounted, size 13A000 bytes (3297 OS/8 blocks) 3248 blocks used
I (4820) [PDP8]: 4 - RKA2 mounted, size 13A000 bytes (3297 OS/8 blocks) 3248 blocks used
I (4830) [PDP8]: 5 - RKB2 mounted, size 13A000 bytes  (3297 OS/8 blocks) 3248 blocks used
I (4840) [PDP8]: 6 - DTA0 mounted, size 048000 bytes ( 756 OS/8 blocks)  737 blocks used
I (4850) [PDP8]: 7 - DTA1 mounted, size 048000 bytes ( 756 OS/8 blocks)  737 blocks used
I (4860) [PDP8]: SD card found, ready  for use
I (4870) [PDP8]: Starting PDP8 Emulator task and telnet server
I (4870) [PDP8]: Starting PDP8 Console task
I (5030) wifi: pm start, type:0

I (6870) [PDP8]: Initializing SNTP
I (6870) [PDP8]: Waiting for system time to be set... (1/10)
I (9870) [PDP8]: Time is set to 17-01-2018 - 11:29:13
I (14910) [PDP8]: Free stack space telnet task is  2212 wordds
I (14910) [PDP8]: Free stack space simulator task is  1388 words
I (14910) [PDP8]: Free stack space console simulator task is  2368 words
I (14920) [PDP8]: Free stack space main task is  3600 words
I (14930) [PDP8]: Free heap space is  40916 bytes
```