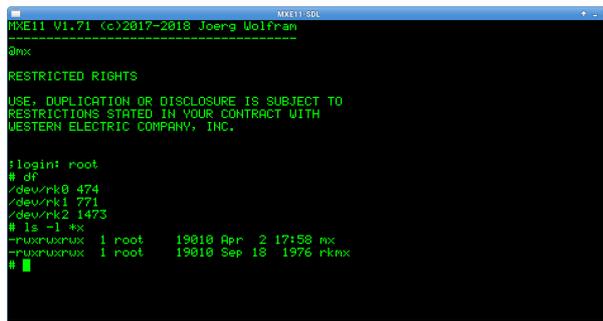


# MXE11: run UNIX on a microcontroller

V1.71 (c) 2017-2018 Joerg Wolfram



```
MXE11 V1.71 (c)2017-2018 Joerg Wolfram
@mx
RESTRICTED RIGHTS
USE, DUPLICATION OR DISCLOSURE IS SUBJECT TO
RESTRICTIONS STATED IN YOUR CONTRACT WITH
WESTERN ELECTRIC COMPANY, INC.

login: root
# df
/dev/rk0 474
/dev/rk1 771
/dev/rk2 1473
# ls -l *x
-rwxrwxrwx 1 root 19010 Apr  2 17:58 mx
-rwxrwxrwx 1 root 19010 Sep 18 1976 rkmx
#
```

## 1 Legal

The programs are subject to the GPL (GNU General Public License) Version 3 or higher, any use of the software/information nonconforming to the GPL or outside the scope of the GPL is prohibited!

The publication of this project is in the hope that it will be useful, but WITHOUT ANY WARRANTY, especially without the implied warranty of MERCHANTABILITY or of INABILIBILITY FOR A PARTICULAR PURPOSE.

The project was developed under Linux, a compatibility with other operating systems is not guaranteed. Likewise, the instructions in this documentation refer to Linux.

All trademarks mentioned in the text are the property of their respective owners. This documentation is derived from the german original so there may be things which are not properly translated.

## 2 Concept

### 2.1 Run Unix on a mikrocontroller?

Looking for an architecture that can be well-emulated with modern microcontrollers and for which there is enough tools and software there, I came across the PDP11. The first approaches have already been made:

<https://dave.cheney.net/tag/pdp11>

but in the end that was not optimal from my point of view. The core problem and bottleneck I see there is the external SPI RAM, but without that you will probably not get out. Or is it?

Mini Unix was developed in the 70s by Heinz Lycklama at Bell Laboratories. It's a complete Unix V6 system, which was modified to run on only 56KBytes of RAM. And above all, no MMU is needed. Roughly enough, 64K of RAM in the controller should suffice, and there are some devices in this size class.

The original project was developed on a SPC56EL60 (ST, PowerPC architecture), in addition there is an SDL port for the PC (actually two) and now some STM32 variants. Since I wanted to be more specific about the matter, I did not use an existing code (like SIMH) for the emulator, but I wrote it from scratch. Also with a view to porting parts to ASM. I have already done this on the SPC56EL60, at the moment I do not have the time to do this for the ARM variants.

### 2.2 What will be emulated?

MXE11 emulates a PDP11 with 56K RAM (28 KWords) without MMU. EIS is implemented, but not yet an FPU. The First and foremost, I did not want to postpone the release too much. A 60 HZ timer (KL11) is available as well as 2 serial interfaces. These are equipped at the emulator level with 256 byte buffers each and work by default with 38400 baud. In a later version, there will possibly also be a baud rate switch, as far as that is of interest.

Compared to a „normal“ PDP11, the interrupt vectors of all serial interfaces are located at the same address. Device switching via the PSW value is performed by the emulator. Main reason is that the second serial Interface was added later and so extensive modifications in the emulator could be avoided. The mass storage is an RK11 interface with 3 RK05 drives emulated, the 3 images are in a single image file accommodated in either a file (PC), on an SD card or an AT45DB642 Dataflash can be located. The data transfer takes place contrary to the original not over DMA, so that during the transfer the CPU emulation is stopped.

In addition, a simple real-time clock (RTC) based on ATMega can be connected to the system time at startup adjust. Without them, the RTC-DATA pin should be left unconnected, then, as with the original kernel (rkmx), the last modification time of the superblock is used as the system time.

**Hint:** Unix V6 can not handle dates after 1999 properly. Therefore, I currently use dates 1978 (ie exactly 40 years ago), but this leads to incorrect day of week data.

Basically, I assume that the emulation is not complete, for instance it was not possible to boot RT11. But for the initial goal, to bring Unix on a mikrocontroller, it seems to be enough.

### 2.3 Install the PC-version

In the **bin** archive are 2 PC versions. They differ only in the resolution in the Text window, **pdp11sim-sdl** displays 24 lines of 80 characters, **pdp11sim-sdl320** only 53 characters per line. This emulates a 320x240 display (6x10 font), which I later use for a mobile version of the emulator would like to. The two programs can be started directly in the directory or even after e.g. /usr/local/bin to be copied. However, it is important that the image file is in the current directory at startup. Overall, the image file contains 3 disk images. All images fit in 8 megabytes and thus on the AT45DB642 Dataflash. Alternatively, an SD card can be used on which the image file is written.

### 2.4 Install the uC-versions

Since SD cards should be more readily available than the AT45DB642, the precompiled binaries are all with SD card support been created. On the **Hardware** Page, the predefined pins are specified. At least, only the appropriate hex file

(.s37, Motorola S-Record Format) must be flashed, Shadow Flash or option bytes remain in their original state. For this, You can use my **UPROG2** or any other suitable programmer.

The second step is to copy the image file to the SD card. This is done via dd, a corresponding script is in the directory. The image file is raw copied to the beginning of the SD card, any existing partitions or data will be overwritten!

For testing, a terminal (for example, serial to USB converter and Minicom on the computer) should be connected to TTY8 be, the baud rate is set to 38400. At startup, the Drive LED lights up and the terminal turns out Version info output. If everything works so far, You will find at the **Manual** page more information about the emulated machine.

## 2.5 Adapt and Modify it

The **src** archive contains the source code of the emulator. Here are also different scripts and created makefiles, which should simplify the compilation as much as possible. The configuration is done via the File **inc/board.h**, here you can set the used pins etc. The abstraction to the controller hardware This is done via a library (unilib) developed by me which is not yet published at the present time is. More details can be found in the header files in the directory **unilib**.

To compile it may be necessary to modify the Makefiles (**TOOLPREFIX**), the libraries needed are all included in the project. On request, I can also create hex files for other configurations, tests as possible.

In order to copy files into the image and out again, You will need to compile and install **u6-fsutil** from the BK-Unix project

<https://sourceforge.net/projects/bkunix/>.

## 2.6 Further Informations

In the **src** archive there is a directory „images” in which the mini Unix original images together with the already carried out modifications. Likewise there is a **boot.ini** with which the system under SIMH can be tested. There, the „rkmx” kernel must be started otherwise the emulated system crashes. In addition, there is a **extdoc** archive, in which I have taken some useful information about the old Unix versions and commands.

## 2.7 Speed/Benchmarks

553/5000 To gauge the usefulness of the emulation and find potential for optimization, benchmarks are sometimes useful. Unfortunately, there is little useful information, with a meaningful speed comparison can do, besides I do not have PDP11 for self-comparison and SIMH throttling is too inaccurate in my opinion. Whetstone needs floating-point support, and I did not get Dhrystone compiled. Two little ones I then found benchmarks, along with comparison values.

### 2.7.1 MIPS

A fairly simple test is to determine the execution time of a given command and this as a relation to take. For this purpose, a single command is executed very often in succession (in the case of the concrete test **INC R4**), measured the execution time and calculated the MIPS value from it.

This is only a comparative value to processors with the same architecture and can be not compare with the VAX-MIPS (at least not directly).

The executable is located at **/usr/bin/mips**, its useful to start it with

```
time mips -o 100
```

and divide 100 by the user seconds to determine the MIPS (million operations per second). The program The displayed MIPS value is usually a bit smaller because it uses the total time to calculate and not the only time claimed by the program. Referred to [mercury.lcs.mit.edu/jnc/tech/V6Unix.html](http://mercury.lcs.mit.edu/jnc/tech/V6Unix.html)

a PDP11/70 reaches approx 3 MIPS. From spec a PDP11/40 has approx 1.01 MIPS, I took this value as a basis for comparison for the table below.

Controller	Takt	MIPS	Comparsion to PDP11/40
SPC56EL60	120 MHz	1,36	134%
SPC56EL60 (ASM)	120 MHz	2,84	282%
STM32F107	72 MHz	—	—%
STM32F103	120 MHz	0,79	78%
STM32F411	96 MHz	—	—%
STM32F405	168 MHz	1,96	194%
STM32L475	78 MHz	0,92	91%

The STM32F411 and STM32F107 have no results, at the time these boards are currently not available to me.

## 2.7.2 Hanoi

Finally, I have yet an indication of the Hanoi benchmark (move 10 discs) found and this was easy (slightly modified) to compile and start.

The executable is located at `/usr/bin/mips`, its useful to start it with

```
time hanoi 1000
```

and divide 1000 by the user seconds to determine the LPS (loops per second).

Referred to

<https://wfjm.github.io/home/w11/impl/performance.html>

the PDP11/53+ reaches a speed of 12,4 LPS, I took this value as a basis for comparison for the table below.

Controller	Takt	LPS	Comparsion to PDP11/53+
SPC56EL60	120 MHz	27,7	223%
SPC56EL60 (ASM)	120 MHz	51,5	415%
STM32F107	72 MHz	10,6	85%
STM32F103	120 MHz	18,3	148%
STM32F411	96 MHz	21,4	173%
STM32F405	168 MHz	34,2	276%
STM32L475	78 MHz	15,4	124%

The not inconsiderable increase in speed through partial assembler code surprises something but the entire code has already been designed for later assembler optimization.

### **3 To Do**

Of course, there is still a lot of optimization and expansion potential, I can think of this:

- FPU-Emulation
- A/D and D/A-converter (PWM)
- digital I/O
- Network via CAN
- ASM-optimizing for STM32
- support for other controller families
- Extended tool to access the file system in the image
- Other programs in the emulator, e.g. Full screen text editor

Some things are in preparation (FPU, I / O), other things are for me currently of low priority (CAN network).

### **4 Changelog**

**May 25th, 2018** version 1.71

- Initial public release