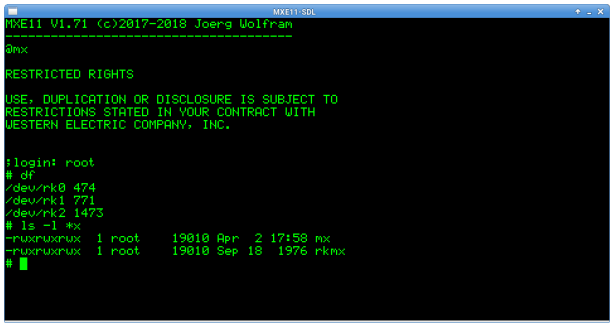
MXE11: Unix auf dem Mikrocontroller

V1.71 (c) 2017-2018 Jörg Wolfram



1 Rechtliches

Die Programme unterliegen der GPL (GNU General Public Licence) Version 3 oder höher, jede Nutzung der Software/Informationen nonkonform zur GPL oder ausserhalb des Geltungsbereiches der GPL ist untersagt!

Die Veröffentlichung dieses Projekts erfolgt in der Hoffnung, daß es Ihnen von Nutzen sein wird, aber OHNE IRGENDEINE GARANTIE, auch ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK.

Das Projekt wurde unter Linux entwickelt, eine Kompatibilität zu anderen Betriebsystemen ist nicht garantiert. Ebenso beziehen sich die Anleitungen in dieser Dokumentation auf Linux.

Alle im Text genannten Marken sind Eigentum des entsprechenden Inhabers.

2 Konzept

2.1 Unix auf dem Mikrocontroller?

Auf der Suche nach einer Architektur, die man mit modernen Mikrocontrollern gutb emulieren kann und für die es genügend Tools und Software gibt, bin ich auf die PDP11 gestossen. Erste Ansätze gab es schon:

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

aber letzendlich war das aus meiner Sicht nicht optimal. Als Kernproblem und Flaschenhals sehe ich hier das externe SPI-RAM, aber ohne das kommt man wohl nicht aus. Oder doch?

Mini-Unix wurde in der 70ern von Heinz Lycklama bei den Bell Laboratories entwickelt. Es ist ein komplettes Unix-V6 System, welches dahingehend modifiziert wurde, auch mit 56K RAM zu laufen. Und vor allen Dingen wird keine MMU benötigt. Grob geschätzt sollten 64K RAM im Controller ausreichen, und in dieser Größenklasse gibt es ja einige.

Das ursprüngliche Projekt wurde auf einem SPC56EL60 (ST, PowerPC Architektur) entwickelt, zusätzlich gibt es einen SDL-Port für den PC (eigentlich zwei) und inzwischen einige STM32 Varianten. Da ich mich etwas genauer mit der Materie befassen wollte, habe ich für den Emulator keinen bestehenden Code (z.B. SIMH) verwendet sondern das Ganze von Grund auf neu geschrieben. Auch mit Blick darauf, Teile nach ASM zu portieren. Beim SPC56EL60 habe ich dies schon getan, bei den ARM Varianten fehlt mir momentan die Zeit dazu.

2.2 Was wird emuliert?

Emuliert wird eine PDP11 mit 56K RAM (28 KWorte) ohne MMU. EIS ist implementiert, eine FPU zur Zeit aber noch nicht. Das hat in allererster Linie Zeitgründe, ich wollte das Release nicht allzuweot hinausschieben. Ein 60 HZ Timer (KL11) ist vorhanden sowie 2 serielle Interfaces. Diese sind auf Emulatorebene mit jeweils 256 Bytes großen Puffern ausgestattet und arbeiten standardmäßig mit 38400 Baud. In eine späteren Version wird es ggf. auch eine Baudratenumschaltung geben, soweit das von Interesse ist.
Gegenüber einer „normalen" PDP11 liegen die Interrupt-Vektoren aller seriellen Interfaces auf derselben Adresse, die Device-Umschaltung über den PSW-Wert wird vom Emulator vorgenommen. Hauptgrund dafür ist, dass das zweite serielle Interface erst später dazugekommen ist und sich so umfangreichere Umbauten im Emulator vermeiden ließen.
Als Massenspeicher wird ein RK11 Interface mit 3 RK05 Laufwerken emuliert, die 3 Images sind dabei in einer einzigen Imagedatei untergebracht, die sich wahlweise in einer Datei (PC), auf einer SD-Karte oder einem AT45DB642 Dataflash befinden kann. Die Datenübertragung erfolgt im Gegensatz zum Original nicht über DMA, so dass während der Übertragung die CPU-Emulation gestoppt wird.
Zusätzlich kann eine einfache Echtzeituhr (RTC) auf ATMega-Basis angeschlossen werden, um die Systemzeit beim Start einzustellen. Ohne diese sollte der RTC-DATA unbeschaltet bleiben, dann wird wie beim Original-Kernel (rkmx) die letzte Änderungszeit des Superblocks als Systemzeit genutzt.
Hinweis: Unix V6 kann nicht mit Datumsangaben nach 1999 umgehen. Daher nutze ich momentan Datumsangaben aus 1978 (also von vor genau 40 jahren), was aber zu falschen Wochentagsangaben führt.
Grundsätzlich gehe ich davon aus, dass die Emulation nicht komplett ist, da es mir z.B. nicht möglich war, RT11 zu booten. Für das eigentliche Ziel, Unix auf einen Mikrocontroller zu bringen, reicht es aber aus.

2.3 Installation der PC-Version

Im **binary** Archiv befinden sich u.a. die 2 PC-Versionen. Sie unterscheiden sich nur in der Auflösung im Textfenster, **pdp11sim-sdl** zeigt 24 Zeilen zu 80 Zeichen an, **pdp11sim-sdl320** nur 53 Zeichen je Zeile. Damit wird ein 320x240 Display (6x10 Font) emuliert, welches ich später für eine mobile Variante des Emulators nutzen möchte. Die beiden Programme können direkt im Verzeichnis gestartet oder auch nach z.B. /usr/local/bin kopiert werden. Wichtig ist aber, dass sich die Image-Datei beim Start im aktuellen Verzeichnis befindet.
Insgesamt enthält die Imagedatei 3 Diskimages. Alle Images passen in 8 Megabytes und damit auf den AT45DB642 Dataflash. Alternativ kann eine SD-Karte genutzt werden, auf die die Imagedatei geschrieben wird.

2.4 Installation der Mikrocontroller-Versionen

Da SD-Karten leichter verfügbar sein sollten als der AT45DB642, sind die vorcompilierten Binaries alle mit SD-Karten-Support erstellt worden. Auf der **Hardware** Seite sind die vordefinierten Pins angegeben. Letztendlich muss nur das

passene Hexfile (.s37, Motorola S-Record Format) geflasht werden, Shadow Flash bzw. Option-Bytes verbleiben im Urzustand. Dazu kann z.B. der von mir entwickelte **UPROG2** Programmer verwendet werden.

Als zweiter Schritt muss die Image-Datei auf die SD-Karte kopiert werden. Das geschieht via dd, ein entsprechendes Script befindet sich mit im Verzeichnis. Die Image-datei wird dabei an den Anfang der SD-Karte kopiert, etwaige Partitionen oder Daten werden dadurch überschrieben!

Zur Inbetriebnahme sollte an TTY8 ein Terminal (z.B. seriell nach USB Wandler und Minicom auf dem Computer) angeschlossen sein, die Baudrate ist auf 38400 einzustellen. Beim Start leuchtet die Drive-LED auf und über das Terminal wird die Versionsinfo ausgegeben. Wenn bis hierhin alles soweit funktioniert, finden sich unter **Anleitung** die nächsten Schritte.

2.5 Anpassen/Modifizieren

Im **src** Archiv befindet sich der Quelltext des Emulators. Hier sind auch verschiedene Scripte und makefiles angelegt, die das Compilieren weitestgehend vereinfachen sollen. Die Konfiguration erfolgt über die Datei **inc/board.h**, hier lassen sich verwendete Pins etc. einstellen. Die Abstraktion zur Controller-Hardware hin erfolgt über eine von mir entwickelte Library (unilib), die zum gegenwärtigen Zeitpunkt noch nicht veröffentlicht ist. Näheres dazu lässt sich aber aus den Header-Dateien im Verzeichnis **unilib** entnehmen.
Zum Compilieren müssen ggf. die Makefiles angepasst werden **(TOOLPREFIX)**, die notwendigen Bibliotheken sind mit im Projekt enthalten. Auf Nachfrage kann ich auch Hexfiles für andere Konfigurationen erstellen und ggf. testen.
Um Dateien in das Image und auch wieder heraus zu „schleusen", ist zur Zeit das **u6-fsutil** aus dem BK-Unix Projekt **https://sourceforge.net/projects/bkunix/** erforderlich.

2.6 Weitere Informationen

Im **src** Archiv gibt es ein Verzeichnis „images" in dem sich die Mini-Unix Originalimages nebst der bereits durchgeführten Modifikationen befinden. Ebenso gibt es eine **boot.ini**, mit der das System unter SIMH getestet werden kann. Dort muss allerdings der „rkmx" Kernel gestartet werden.
Zusätzlich gibt es noch ein **extdoc** Archiv, in dem ich ein paar nützliche Informationen zu den Unix-Versionen und Kommandos zusammengetragen habe.

2.7 Geschwindigkeit/Benchmarks

Um die Brauchbarkeit der Emulation abzuschätzen und Optimierungspotential aufzuspüren sind Benchmarks manchmal ganz nützlich. Leider finden sich wenig brauchbare Angaben, mit deen sich ein sinnvoller Geschwindigkeitsvergleich anstellen lässt, außerdem habe ich keine PDP11, um selbst Vergleichswerte zu bestimmen und das Throttling von SIMH ist meiner Meinung nach zu ungenau. Für Whetstone braucht es Fliesskomma-Unterstützung und Dhrystone habe ich nicht compiliert bekommen. Zwei kleine Benchmarks habe ich dann doch gefunden, nebst Vergleichswerten.

2.7.1 MIPS

Ein recht einfacher Test ist, die Ausführungszeit eines bestimmten Befehls zu bestimmen und diese als Relation zu nehmen. Dazu wird ein einzelner Befehl sehr oft hintereinander ausgeführt (beim konkreten Test **INC R4**), die Ausführungszeit gemessen und daraus der MIPS-Wert berechnet.
Dabei stellt dies lediglich einen Vergleichswert zu Prozessoren mit gleicher Architektur dar und lässt sich nicht mit den VAX-MIPS vergleichen (zumindest nicht direkt).

Das fertige Compilat liegt unter **/usr/bin/mips**, zweckmäßigerweise startet man es mit

```
time mips -o 100
```

und teilt 100 durch die user-Sekunden, um die MIPS (million operations per second) zu bestimmen. Der vom Programm angezeigte MIPS-Wert ist meist etwas kleiner, da es die Gesamtzeit zur Berechnung heranzieht und nicht die vom Programm beanspruchte.
Lt. **mercury.lcs.mit.edu/ jnc/tech/V6Unix.html**

erreicht die PDP11/70 etwa 3 MIPS. Laut Spezifikation erreicht eine PDP11/40 1.01 MIPS, diesen Wert habe ich als Vergleichsbasis für die nachstehende Tabelle genommen.

Controller	**Takt**	**MIPS**	**Vergleich 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%

Die Boards mit STM32F107 und STM32F411 hatte ich zum Testzeitpunkt nicht zur Verfügung.

2.7.2 Hanoi

Letztendlich habe ich aber noch eine Angabe zum Hanoi-Benchmark (10 Scheiben) gefunden und dieser ließ sich auch (leicht modifiziert) compilieren und starten.
Das fertige Compilat liegt unter **/usr/bin/hanoi**, zweckmäßigerweise startet man es mit

```
time hanoi 1000
```

und teilt 1000 durch die user-Sekunden, um die LPS (loops per second) zu bestimmen.
Lt.

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

erreicht die PDP11/53+ ganze 12,4 LPS, diesen Wert habe ich als Vergleichsbasis für die nachstehende Tabelle genommen.

Controller	**Takt**	**LPS**	**Vergleich 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%

Die nicht unerhebliche Geschwindigkeitssteigerung durch teilweisen Assemblercode überrascht etwas, aber der gesamte Code wurde schon auf eine spätere Assembler-Optimierung hin ausgelegt.

3 Was noch fehlt

Natürlich gibt es noch einiges Optimierungs- und Erweiterungspotenzial, dazu fallen mir ein:

- FPU-Emulation
- A/D und D/A-Wandler (PWM)
- Digitale I/O
- Netzwerk via CAN
- ASM-Optimierung für STM32
- Unterstützung für andere Controllerfamilien
- Erweitertes Tool zum Zugriff auf das Dateisystem im Image
- Weitere Programme im Emulator, z.B. Fullscreen-Texteditor

Einiges ist in der Vorbereitung (FPU, I/O), andere Dinge sind für mich zur Zeit eher zweitrangig (CAN-Netzwerk).

4 Changelog

25.5.2018 Version 1.71

- erste öffentliche Version