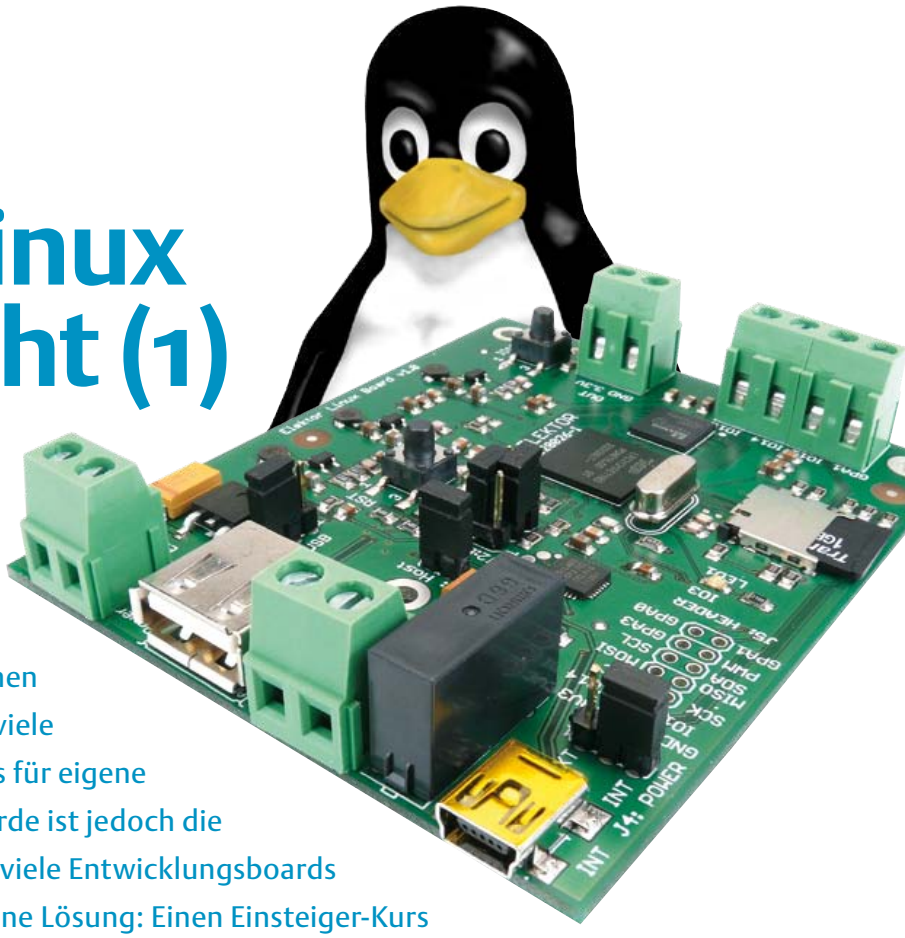
Embedded Linux leicht gemacht (1)

Der Einstieg



Von Benedikt Sauter [1]

Linux läuft heutzutage auf vielen verschiedenen Geräten – sogar in Kaffeemaschinen. Es gibt viele Elektroniker, die am Einstieg in Linux als Basis für eigene Controller-Projekte interessiert sind. Eine Hürde ist jedoch die scheinbar hohe Komplexität, außerdem sind viele Entwicklungsboards recht teuer. Elektor hat für beide Probleme eine Lösung: Einen Einsteiger-Kurs und eine kompakte, preiswerte Platine.

Auf dem Markt gibt es bereits einige Einsteigerkurse für 8-bit-Mikrocontroller, doch noch wenig Literatur oder Webseiten für Embedded-Linux-Anfänger. Die meisten Beschreibungen setzen zu viele Vorkenntnisse voraus und driften zu schnell in kryptische Quelltexte, Spezialthemen und Feinheiten ab. Dabei ist ein Linux-Betriebssystem für Mikrocontroller eigentlich „nur" eine klassische, gut strukturierte und sehr modulare „Firmware". Mit einem soliden Mikrocontroller-Grundwissen kann man auch ein solches scheinbar komplexes System verständlich erklären.

Was benötigt man für einen Einstieg in diese Welt? In den Anfangszeiten von Computern und Mikroprozessoren war es für jeden Interessierten kein Problem, die Hardware, das Betriebssystem, die Anwendungen, Treiber und das gesamte Zusammenspiel eines Computers zu verstehen. Der Grund lag wohl primär darin, dass es noch nicht solch eine enorme Auswahl an Komponenten wie heute gab. Man konnte sich den wenigen vorhandenen Bausteinen und Anwendungen viel intensiver widmen. In der Regel führte man den Aufbau der Hardware wie auch die Inbetriebnahme in Eigenregie durch. Um Fehlern überhaupt auf den Grund gehen zu können, musste man logischerweise ganz genau verstehen, wie das System arbeitet.

Auf diese Art und Weise soll Linux auch in dieser Artikelserie erarbeitet werden. Als Hardware nutzen wir ein kompaktes Board, das mit allem ausgestattet ist, was man für ein modernes Embedded-Projekt benötigt (**Bild 1**), darunter USB, SD-Karten-Anschluss und verschiedene Erweiterungsmöglichkeiten. Auch der Anschluss an ein Ethernet-Netzwerk ist kein Problem, hierzu mehr in einer kommenden Folge des Kurses. Basis des Elektor-Linux-Boards ist das Open-Source-Projekt „Gnublin", das für Ausbildungszwecke an der Hochschule Augsburg entwickelt wurde [2].
Das Linux-Board kommt bewusst ohne Spezial-Bausteine aus. Die zweilagige Platine wird von Elektor fertig bestückt angeboten (siehe **Bild 2**). Auf den Schaltplan gehen wir ausführlich im zweiten Teil ein; der Vollständigkeit halber zeigen wir ihn aber schon jetzt (**Bild 3**). Die Hardware steht unter der „freedomdefined.org/OSHW"-Lizenz, was beinhaltet, dass wir auch die CAD-Daten zur Verfügung stellen [3]. Selbstverständlich ist auch die Software zu diesem Projekt komplett open source, sie kann wie immer von der Elektor-Website heruntergeladen werden [3].

Schritt für Schritt

Eine kleine Roadmap unseres Kurses sieht man in **Bild 4**. Der Linux-Einsteiger sollte zuerst einmal verstehen, wo die wichtigsten Anwendungen und Softwarekomponenten ihren Ursprung haben. Eine Reihe dieser Komponenten bilden die Basis für unser Elektor-Linux (oder andere aktuelle Linux-Systeme, z.B. für den PC). Außerdem lernt man, wie die Hardware aufgebaut ist und funktioniert. Anschließend nehmen wir uns vor, eine passende Linux-Entwicklungsumgebung auf dem PC zu installieren, um Quelltexte selbst übersetzen zu können. Denn wenn man Linux auf einem Mikrocontroller einsetzen möchte, sollte auch die PC-Entwicklungsumgebung unter Linux laufen (schon allein wegen der Pfadnamen). Windows-User können Linux zum Beispiel in einer virtuellen Maschine installieren.

Anschließend erhält man durch praktische Beispiele nach und nach eine bessere Vorstellung, wie das Betriebssystem Linux funktioniert. Unser Abschlussprojekt wird eine einfache Heizungs-Regelung sein, mit einer grafischen Anzeige zur Konfiguration und Daten-Auswertung in einem Browser.

Der Startschuss von GNU und Linux

Jeder angehende Linux-Profi sollte wissen, warum und wie die freie Unix-Implementierung GNU/Linux entstanden und organisiert ist. Dies ermöglicht auch eine bessere Erkennung der Grenzen dieses Betriebssystems. Vor allem bei technischen Problemen

sollte man erkennen, welche Komponente oder Software die Ursache eines Fehlers sein könnte.

Begonnen hat die Entwicklung von UNIX [4] im Wesentlichen 1969, und zwar an den Bell Laboratories in den USA. Ken Thompson erstellte die erste UNIX-Version in Assembler. Um zu testen, was an Schnittstellen und Treibern benötigt wurde, schrieb Ken Thompson gemeinsam mit Dennis Ritchie das Spiel „Space Travler“. Diesen Schritt kann man sich als Mikrocontroller-Programmierer sehr gut vorstellen, denn jeder Embedded-Entwickler muss zunächst planen, wie die Software (inklusive Hardwareansteuerung und Hilfsfunktionen) zu Gunsten einer hohen Wiederverwendbarkeit der Quelltexte strukturiert werden soll. Ken und Dennis wussten bald, welche Komponenten man für ein Betriebssystem benötigt, und wie man das Ganze organisieren muss. Schließlich schrieben sie den Kern des Betriebssystems in den Jahren 1972 bis 1974 neu, und zwar in der Sprache C, die ebenfalls hier ihren Ursprung hatte. Das Betriebssystem wurde kostenfrei inklusive C-Compiler an Universitäten verteilt.

Erst Ende der 1970er Jahre merkte AT&T - Betreiber der Bell Laboratories - dass sie UNIX gewinnbringend vermarkten konnten. Bis zu dieser Zeit war es selbstversändlich, dass man Software verteilte und austauschte. Keiner hatte irgendwelche „Piraterie-Gefühle“ oder machte „illegale Sachen“. Es wurde gerne getauscht; mit dem Ziel, gemeinsam immer bessere Versionen der Software zu erhalten. Diese grundlegende Einstellung ist immer noch zentraler Kern von freier Software und Open-Source-Software [5][6].

Nachdem aber AT&T begann, UNIX zu verkaufen, durfte es nicht mehr frei getauscht werden. Die Verwendung in Vorlesungen oder zum Selbststudium war wegen der hohen Lizenzkosten plötzlich nicht mehr möglich. In dieser Zeit begannen immer mehr Firmen, eigene UNIX-Versionen und Varianten zu lizenzieren. So entstand z.B. SINIX von Siemens, das seinen Ursprung in der UNIX-Version Xenix von Microsoft hatte. Dass UNIX primär nur noch für Firmen verfügbar war, ließ einem Mitarbeiter des amerikanischen MITs [7], Richard Stallman, keine Ruhe. Schön war die Erinnerung an

Eigenschaften des Elektor-Linux-Boards

- 2-Lagen-Board und leicht erhältliche Bauteile
- Keine Spezialhardware (Debugger, Programmer, etc.) notwendig
- Bootet vollständig von einer SD-Karte
- Fertig installiertes Linux
- 180 MHz, 8 MB RAM (32 MB möglich), 64 MB Swap-Speicher
- Integrierter USB-RS232-Wandler als Konsole
- Relais, externe Versorgung und Taster für schnelle Tests
- On-board 4 x GPIO, 3 x A/D-Kanäle und PWM-Kanal
- I2C und SPI als Schnittstelle aus Linux heraus nutzen
- USB-Schnittstelle für Erweiterungskarten

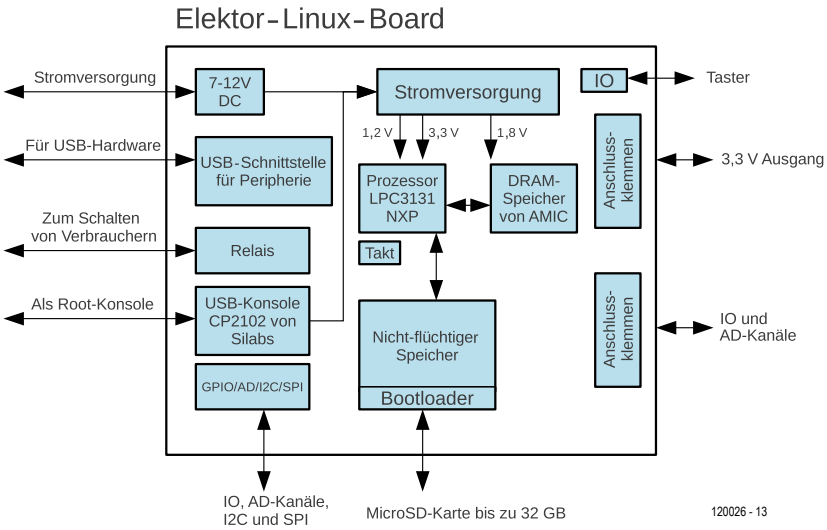


Bild 1. Das Board ist eine mächtige Basis für eigene Controller-Projekte, auch ein Netzwerkzugang ist möglich.

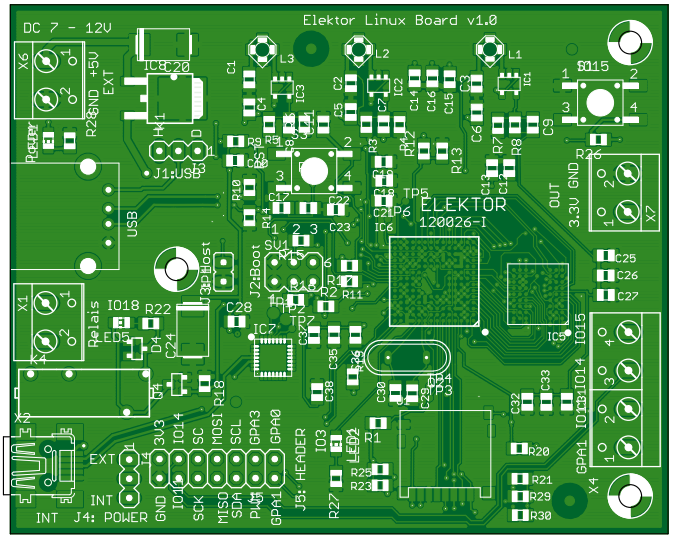


Bild 2. Die Platine wird von Elektor fertig bestückt angeboten.

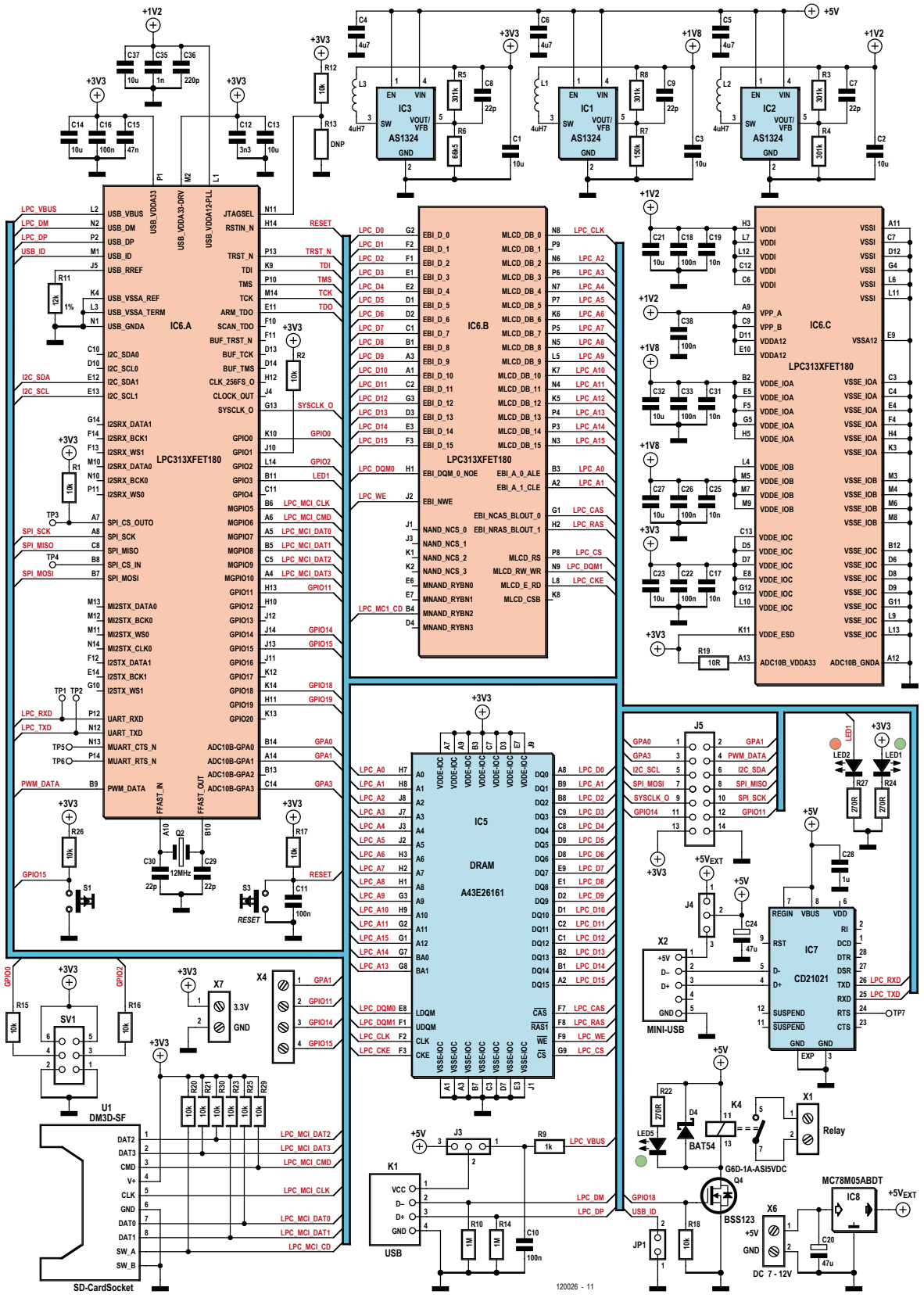


Bild 3. Für ein solch leistungsfähiges Board ist der Schaltplan sehr übersichtlich.

die Zeit, in der man UNIX mit Kollegen und Freunden tauschen konnte. Es gab nur eine Lösung - es musste ein komplett freies UNIX-System neu entwickelt werden!
Die Geburtsstunde von GNU [8] („*GNU's Not Unix*“) war 1983. Vor Richard Stallman lag eine Menge Arbeit: Es musste ja alles neu implementiert werden, um ein zu 100 % freies Betriebssystem zu haben.
Benötigt wurden:

- C-Compiler, Linker, Assembler (Toolchain)
- ein Text-Editor, um Quelltexte schreiben zu können
- Betriebssystem-Kernel
- Diverse Hilfsprogramme
- Root-Dateisystem des Betriebssystems

1990 waren schließlich alle wesentlichen Teile entwickelt - mit Ausnahme des Betriebssystem-Kernels. Richard Stallman wusste: Erst wenn der Texteditor und auch der Compiler stabil funktionierten, machte ein Einstieg in die Kernel-Entwicklung Sinn.

Die Anfänge von Linux

Etwa zur gleichen Zeit kaufte sich ein finnischer Student namens Linus Torvalds seinen ersten x86-Computer und entwickelte ein einfaches Terminal-Programm, um seinen Computer besser zu verstehen [9]. Für die Entwicklung setzte er Minix ein, das von einem Professor und seinem Team aus Amsterdam entwickelt wurde. Dieses Unix-Derivat war noch bezahlbar (und ist bis heute im Einsatz). Nach und nach merkte Linus Torvalds, dass sich sein Terminal-Programm immer mehr hin zu einem Betriebssystem entwickelte. Um auf ein großes Archiv an Software zurückgreifen zu können, war ihm klar, dass er ein POSIX-kompatibles System schaffen musste. POSIX definiert einen Standard, wie ein UNIX Betriebssystem auszusehen hat. In der Bücherei stand (zu unser aller Glück) die passende Literatur für POSIX-Systeme: Vermutlich war es ein Handbuch von einem anderen der vielen UNIX-Derivate. Im Wesentlichen musste Linus Torvalds ja nur wissen, wie die Systemaufrufe hießen bzw. welche Parameter übergeben werden mussten.
1992 stellte der junge Entwickler seine Errungenschaft im Internet zum freien

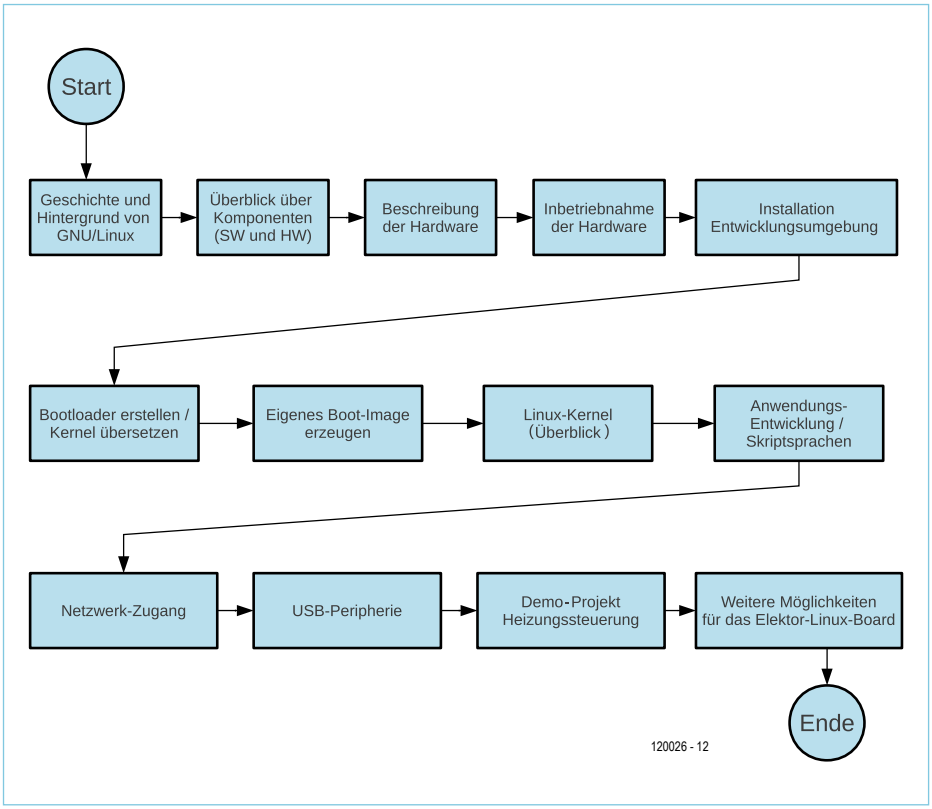


Bild 4. Roadmap unseres mehrteiligen Embedded-Linux-Einstiegs.

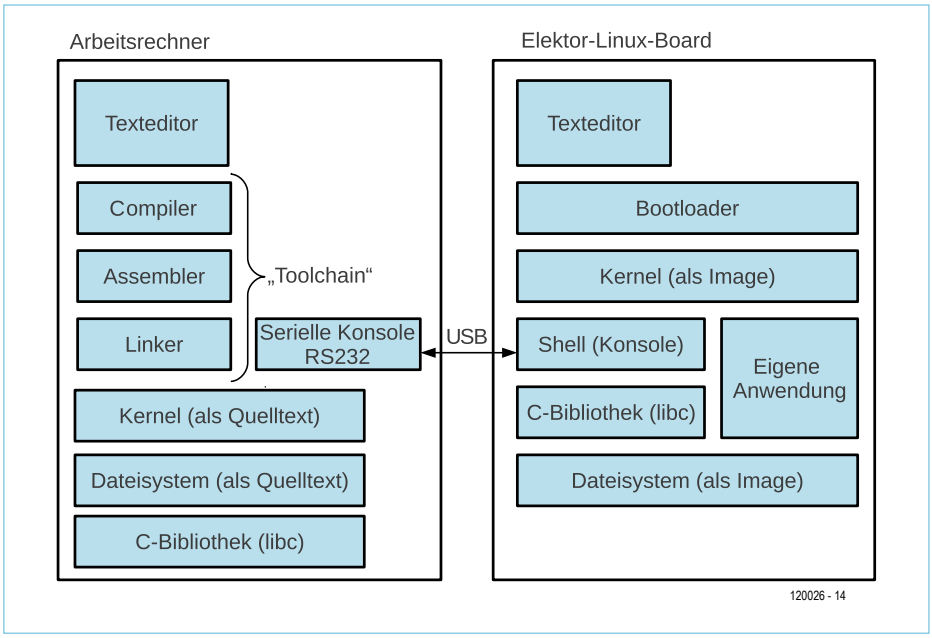


Bild 5. Bei der Software-Erstellung spielen ein (Linux-)Entwicklungsrechner und das Ziel-System zusammen.

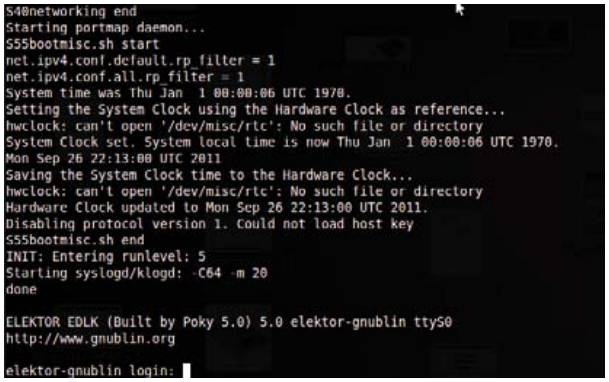


Bild 6. Screenshot von der Konsole.

Download zur Verfügung [10]. Auf der Suche nach einer passenden Quelltext-Lizenz traf es sich gut, dass er zufällig einen Vortrag von Richard Stallman an der Uni hörte. Die GNU GPL (die Open-Source-Lizenz des GNU-Projekts) passte sehr gut. Und jetzt passierte etwas, was nie geplant war: Die Open-Source-Community erkannte schnell, dass der Kernel Linux von Linus Torvalds genau der fehlende Teil im GNU-Projekt von Richard Stallman war! Zwar hatte Stallman schon mit einem freien GNU-Kernel unter dem Namen Hurd begonnen – dieser erlangte aber nie die Bedeutung wie das Stück Software aus Finnland. Erst durch die Komplettierung des GNU-Projekts mit dem Linux-Kernel war ein komplett freies und offenes Betriebssystem möglich geworden. Der Fairness halber sollte man eigentlich nicht nur von Linux, sondern von GNU/Linux sprechen. Denn Linux bezeichnet wirklich nur den Kernel, alles andere kommt vom GNU-Projekt.

Ein grober Überblick

Nach diesem kurzen Blick zurück wollen wir uns nun einen groben Überblick über die benötigten Komponenten unseres GNU/Linux-Systems verschaffen (siehe **Bild 5**). Grundsätzlich hat sich seit den Anfängen nichts geändert. Die grundlegenden Anwendungen von damals benötigen wir auch heute noch für unser Elektor-Linux-Board.
Im Rahmen dieses Projekts werden wir einige dieser Ur-Programme verwenden. Um einige werden wir aber auch gezielt einen Bogen machen. Der Grund wird in den entsprechenden Abschnitten genannt.

Texteditor

Der moderne Entwickler greift gerne auf einen Texteditor seiner Wahl mit Syntax-Highlighting, automatischer Code-Vervollständigung und integrierter API-Dokumentation zurück. Um schnelle Änderungen direkt auf dem Elektor-Linux-Board vornehmen zu können, benötigt man einen Text-Editor, der einfach von der Linux-Konsole (siehe unten) aus zu verwenden ist.
Es gibt klassische Editoren wie „vi“ (oder „vim“ in der benutzerfreundlicheren Version) oder „nano“, die hierfür perfekt geeignet sind. Beide befinden sich im Root-Dateisystem (siehe unten) des Elektor-Linux-Boards. Der klassische Linux-Entwickler verwendet den gleichen Text-Editor auch auf seinem Arbeitsrechner, so muss man nicht umdenken.

Darüber hinaus gibt es noch den sehr bekannten Editor „Emacs“. Eventuell hat der ein oder andere von diesem Editor bereits gehört. Er wurde von Richard Stallman im Rahmen des GNU-Projekts entwickelt. Emacs eignet sich wegen seines großen Funktionsumfangs sehr gut für Profis; Anfänger tun sich mit einem einfacheren Editor leichter.

Compiler + Linker + Assembler = Toolchain

Um Programme auf einem Prozessor zum Laufen zu bringen, benötigt man Maschinencode für die entsprechende Zielarchitektur. Die GNU-Toolchain beinhaltet alle Softwarekomponenten, um die Umwandlung von C nach Maschinencode erfolgreich durchführen zu können. Sie wurde dahingehend entwickelt, dass man einfach neue Maschinenbefehlssätze hinzufügen kann. So gibt es Versionen für x86, amd64, AVR, ARM, MIPS, MSP430 und viele weitere. Da das Elektor-Linux-Board mit einem ARM-kompatiblen Mikrocontroller ausgerüstet ist, werden wir eine entsprechende ARM-Toolchain verwenden. Dazu aber mehr, wenn wir sie installieren.

Kernel

Der Kernel ist der Kern des Betriebssystems. Ursprünglich stammt der Quelltext von Linus Torvalds, mittlerweile arbeiten einige 10.000 Kernel-Entwickler an den Quelltexten mit. Torvalds hat aber immer das letzte Wort; Veränderungen und Erweiterungen nimmt er an oder lehnt sie ab. Wer damit nicht einverstanden ist, kann natürlich seinen eigenen Kernel pflegen, es ist ja alles open source. Bis heute gibt es jedoch keine nennenswerte Abspaltung des Linux-Kernels. Die Softwareentwicklung wird über Mailinglisten organisiert. Jeder kann Mitglied in den Listen werden und Vorschläge einreichen. Dort wird die Änderung dann von vielen anderen bewertet und diskutiert. Der Kernel ist bis auf ein paar wenige Zeilen vollständig in C geschrieben. Er kann einfach mit der GNU-Toolchain von C nach Maschinencode übersetzt werden. Dies werden wir natürlich dann auch in der Artikelserie an entsprechender Stelle machen.

Dateisystem

Unter Windows ist es selbstverständlich, Dateien in den Ordner „Dokumente und Einstellungen“ des Benutzers zu legen. Programme werden im Ordner C:\Programme installiert, betriebssystemnahe Dateien liegen im Ordner „System32“ unter C:\Windows. Windows hat eben wie jedes andere

Betriebssystem eine Struktur, um Ordnung in die vielen Programme und Dateien zu bekommen. Nun stellt sich natürlich die Frage, woher das Dateisystem bei Embedded GNU/Linux stammt. In diesem Fall nicht von Linus Torvalds oder Richard Stallman; eine gemeinsame Basis für jedes Unix/Linux hat sich vielmehr durch den POSIX-Standard Schritt für Schritt etabliert. Mehr Bedeutung haben die so genannten Root-Dateisysteme später durch bekannte Distributionen wie Debian, Suse etc. erhalten. Diese Distributionen stellen dem Anwender ein fertiges GNU/Linux-System mit installierten Anwendungen, grafischen Oberflächen und einem aktuellen Kernel zur Verfügung. Um auf unserem Board Linux nutzen zu können, benötigt man ebenso solch ein Root-Dateisystem. Meistens genügt ein sehr kleines mit einem geringen Umfang an Programmen und Bibliotheken. Ein klassisches Desktop-Linux wäre für ein Embedded-Board viel zu umfangreich. Es gibt spezielle Programme, mit denen man sich ein angepasstes Root-Dateisystem selbst erstellen kann. Alternativ gibt es alle großen Distributionen auch schon für ARM-Prozessoren; dazu mehr an entsprechender Stelle.

C-Bibliothek

Ein Computer oder Produkt lebt immer von den Anwendungen. Das Betriebssystem ist ja nur die Instanz im Hintergrund, die es ermöglicht, Hardware zu nutzen, Speicher zu reservieren, Kommunikation über Schnittstellen und Netzwerke zu ermöglichen und vieles mehr. Schreibt man Anwendungen, so möchte man nicht jedes Mal wieder aufs Neue Funktionen für das Dateien lesen und schreiben, String-Funktionen etc. programmieren. Dafür gibt es für alle Entwickler eine C-Bibliothek als Standard, in der bekanntesten Version als „libc“ bezeichnet. Für Embedded-Systeme gibt es diese in noch schlankeren Versionen, denn so viel Speicher und Rechenleistung wie auf einem Arbeitsrechner gibt es hier meist nicht.
Die C-Bibliothek ist die Schnittstelle zwischen Anwendung und Kernel. Zudem liefert die C-Bibliothek viele Hilfsfunktionen, welche immer wieder benötigt werden. Die C-Bibliothek wird dynamisch zur Laufzeit geladen, die Programme werden dynamisch hierauf gelinkt. So spart man eine Menge Speicherplatz, da man die Bibliothek für alle Programme gemeinsam nur einmal im Speicher vorhalten muss.

Serielle Konsole / Shell

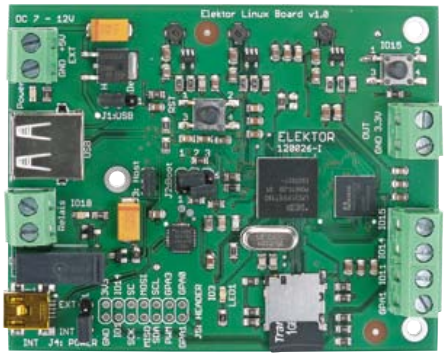
Die Konsole – vergleichbar mit der „Eingabeaufforderung“ bei Windows – nimmt Benutzereingaben entgegen, löst Aktionen aus (auch auf einem entfernten Gerät) und zeigt Ergebnisse an. Die Konsole ist so Schnittstelle in das System. Im Allgemeinen nutzt man eine „Shell“ als Linux-Konsole. Das ist ein eigenes Programm, das viele Hilfsmittel mitbringt, um ein Linux-System einfach bedienbar zu machen. Auf die Shell kommen wir später zurück.

An einem mit Linux gestartetem Rechner ist die klassische Root-Konsole der Bildschirm und die Tastatur (auf klassische Konsole umschalten mit STRG-ALT-F1). Netzwerkadministratoren verwenden typischerweise Protokolle wie SSH oder bei unkritischen Daten auch Telnet. Sie ermöglichen den Zugriff auf die Root-Konsole über eine Netzwerkverbindung. Eine dritte Option ist die Nutzung einer Konsole über eine RS232-Schnittstelle. Als passendes Gegenstück benötigt man auf dem Arbeitsrechner ein Konsolen-Programm, das sich per serieller Schnittstelle verbinden kann (wie z.B. unter Windows Hyperterminal oder TeraTerm bzw. unter Linux z.B. picocom).

Ausblick

Im nächsten Artikel dieser Serie zeigen wir, wie die Hardware (Bild 1) aufgebaut ist. Näher betrachtet werden die Stromversorgung, der Mikroprozessor, der SDRAM-Baustein und die Schnittstellen. Ein weiteres Thema wird der Bootablauf sein. Dank der vorinstallierten Demo-Software (**Bild 6**) kann man direkt nach Erhalt des Boards erste Erfahrungen sammeln.

(120026)



Weblinks

[1] sauter@embedded-projects.net
[2] www.gnublin.org
[3] www.elektor.de/120026
[4] http://de.wikipedia.org/wiki/Unix
[5] http://de.wikipedia.org/wiki/Freie_Software
[6] http://de.wikipedia.org/wiki/OpenSource
[7] http://de.wikipedia.org/wiki/Massachusetts_Institute_of_Technology
[8] www.gnu.org
[9] „Die Software Rebellen“, Glyn Moody, ISBN-3-478-38730-2
[10] www.kernel.org