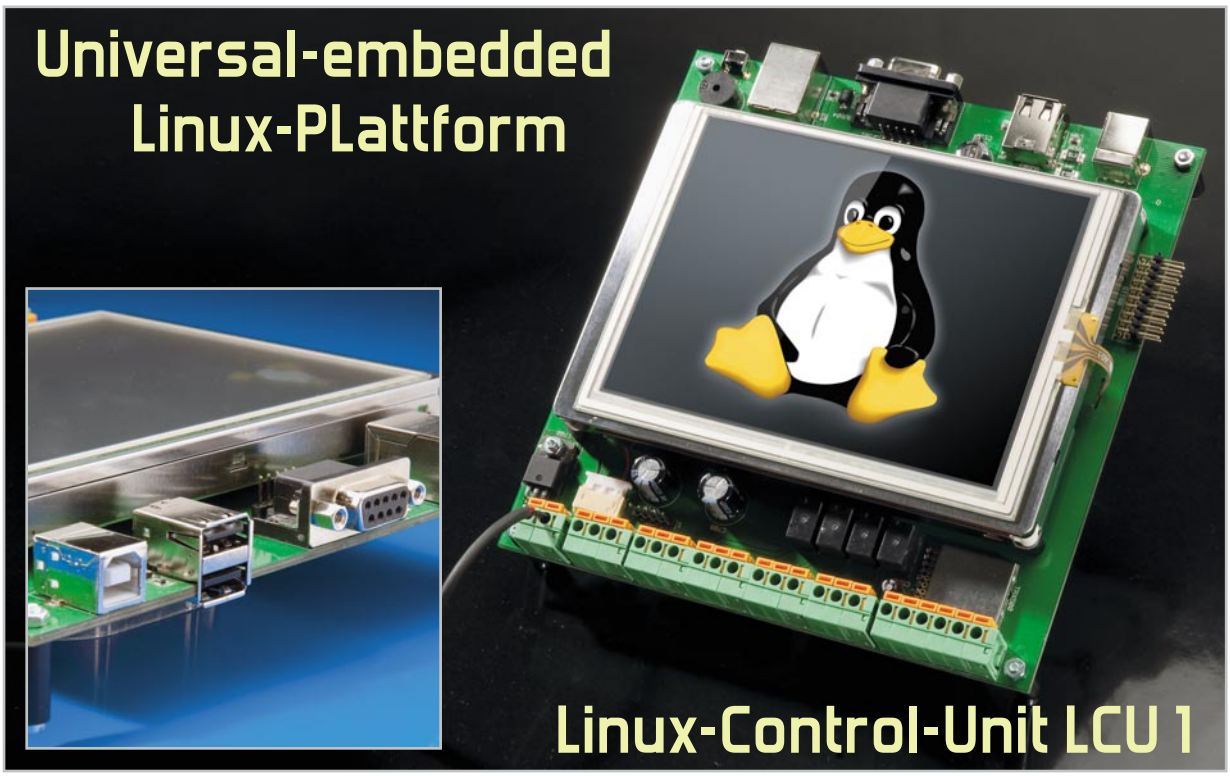
Universal-embedded Linux-PLattform



Mit der Linux-Control-Unit stellen wir eine komplette Hard- und Software-Plattform mit zahlreichen unterschiedlichen Schnittstellen, Eingängen, Ausgängen und Speichermöglichkeiten vor, die jeden, der sich mit dem Thema Embedded-Linux-Systeme beschäftigen möchte, in die Lage versetzt, schnell und preiswert zu einer eigenen kompletten Lösung, vornehmlich im Steuerungsbereich, zu kommen.

Embedded Systems und Linux

Embedded System, ein stehender Begriff im englischen Sprachgebrauch, dem wir mit dem deutschen Begriff „Eingebettetes System" in seiner Bedeutung näherkommen. Als eingebettetes System bezeichnet man ein kleines, autarkes Rechnersystem, das eine bestimmte Aufgabe innerhalb eines anderen technischen Systems erfüllt – meist als Steuerung dieses Systems. Dies beginnt bereits bei einer modernen Waschmaschinensteuerung, geht über die Steuerung und Visualisierung an Info- oder Bedienterminals bis hin zu zentralen Steuerungsaufgaben z. B. in der Gebäudeautomation, in Maschinen oder Geräten, in der Automatisierungs-, Mess- oder Medizintechnik.

Selbst als Steuerung moderner Satellitenempfänger, Videorecorder, Mediaserver usw. verrichten eingebettete Systeme unauffällig ihren Dienst. Ihre Leistungsfähigkeit hat sich in den vergangenen Jahren mit der Verfügbarkeit immer komplexerer, stromsparenderer und leistungsfähigerer Mikroprozessoren bis hin in den 32-Bit-Bereich immer weiter gesteigert, wobei gleichzeitig die Kosten für diese Prozessorsysteme ständig sinken.

Typische Beispiele für entsprechende Prozessorfamilien sind heute die ARM-Prozessorarchitekturen, auch Intelx86-Architekturen und FPGA-Systeme sind vertreten. Speziell die ARM-Systeme sind heute so leistungsfähig, dass sie häufig die technische Grundlage für komplette Betriebssysteme wie Linux oder „Windows Embedded" bilden können. Beispiele hierfür halten wir täglich in der Hand – vom Smartphone über den PDA bis zum DSL-/Netzwerkrouter und dem mobilen Navigationssystem.

Hohe Taktfrequenzen und hohe Datenverarbeitungsbreite sowie die Verwaltung bzw. Implementierung umfangreicher Speichersysteme sichern schnelle, der Aufgabe angemessene Verarbeitungszeiten. Unter diesen Voraussetzungen liegt es nahe, solche kompakten, gegenüber Standard-Prozessoren aus der PC-Technik deutlich Strom und Kosten sparenden Hardware-Plattformen auch mit kompletten Betriebssystemen, wie eben Linux, zu versehen, um auszuführende Aufgaben auch beispielsweise über ein grafisches Bedienkonzept und mit aus der PC-Technik bekannten Schnittstellen ausführen zu können.

Linux bietet sich als Betriebssystem nicht nur deshalb an, weil es über einen sehr schlanken und damit ressourcenschonenden Kernel verfügt und als Open-Source-Projekt kostenlos nutzbar ist. Es liegt ja bekanntlicherweise als offener Quellcode vor und ist damit für den Programmierer leicht an die eigenen Bedürfnisse anzupassen. Unzählige, von einer weltweit riesigen Open-Source-Community entwickelte Standard-Programmbausteine stehen dem Entwickler zum freien Einsatz bzw. zur eigenen Modifikation zur Verfügung, um eine spezielle Aufgabe lösen zu können. Dazu gibt es weltweit enorme geistige Ressourcen über Foren, Wikis, Tutorials, Veröffentlichungen usw.

Die LCU 1

Die LCU 1 ist mit der Prämisse entwickelt worden, dem ambitionierten Software-Entwickler, sei er Einsteiger oder bereits fortgeschrittener Programmierer, eine Hardware-Plattform zur Verfügung zu stellen, die möglichst universell ausgeführt ist und damit alle Optionen für die Entwicklung eigener Anwendungen bietet. Sie kann sowohl als Entwicklungssystem als auch direkt als Embedded-System laufen, z. B. als Haustechnik-Rechner mit integrierter Bedien- und Anzeigeeinheit, der zentral mit Aktoren, Sensoren und Bedienelementen zusammenarbeitet.
Als CPU haben wir zu einer mit 190 MHz getakteten ARM-Prozessorarchitektur von Atmel gegriffen, die sich mit ihrer reichhaltigen Ausstattung mit Schnittstellen und ihrer Leistungsfähigkeit besonders für die zu erfüllende Aufgabe als Entwicklungssystem eignet. Atmel ist einer der Lizenznehmer für die Advanced RISC-Machine (ARM, ursprünglich nach dem Entwickler Acorn auch Acorn RISC Machine genannt). Mehr hierzu findet man unter [1].
Die Architektur des ARM9-Mikrocontrollers bietet eine Menge. So können 32 KByte interner ROM und 160 KByte interner SRAM über ein externes Bus-Interface (EBI) von externem SDRAM, SRAM, NAND-Flash-Speicher und Compact Flash unterstützt werden. Der Bootvorgang kann aus dem internen oder einem externen nicht flüchtigen Speicher erfolgen. Zwei USB-Host-Ports und ein Device-Port sorgen für USB-Anbindung und ein leistungsfähiger LCD-Controller für die komfortable Ansteuerung von Displays. Ein Watchdog und ein Echtzeit-Timer gehören ebenso zur Ausstattung wie die notwendigen Oszillatoren für alle Teile des Mikrocontrollers. Schließlich stehen noch drei 32-Bit-PIOs sowie drei USARTs sowie ein Multimedia-Card-Interface zur Verfügung. Selbst unter Volllast benötigt diese CPU nicht mehr als 100 mA, so ist eine solche Applikation auch gut für tragbare Geräte geeignet. Alles über die Ausstattung und die Möglichkeiten dieser Controller-Familie ist unter [2] zu finden.
Dieser Baustein ist somit die ideale Basis für eine Entwicklungs- und Applikations-Plattform, wie sie die LCU 1 darstellt. Sie ist nur wenig größer als das hier eingesetzte Touchscreen-TFT-Display mit 14,5 cm (5,7") Bildschirmdiagonale. Weitere Details zur Ausstattung besprechen wir in der Schaltungsbeschreibung.

Linux – das freie Betriebssystem

Wer sich mit der Geschichte von Linux befasst, stößt unweigerlich auf die Begriffe „GNU-Projekt", „GPL" und „UNIX". UNIX, ein Mehrplatz-Benutzer-Betriebssystem, entstand bereits in den 1970er Jahren, es war zunächst an Universitäten verbreitet, und bis Anfang der 1980er Jahre war auch der Quellcode des Systems frei zugänglich. Anfang der 1980er Jahre war Schluss mit freiem Quellcode (die letzte freie Version UNIX V7 erschien 1979), einige Firmen wie z. B. AT&T kommerzialisierten das Produkt zu einer proprietären, über Lizenzen zu bezahlenden Software. Damit konnten sich einige Wissenschaftler, namentlich Richard Stallman vom bekannten Massachusetts Institute of Technology (MIT), nicht

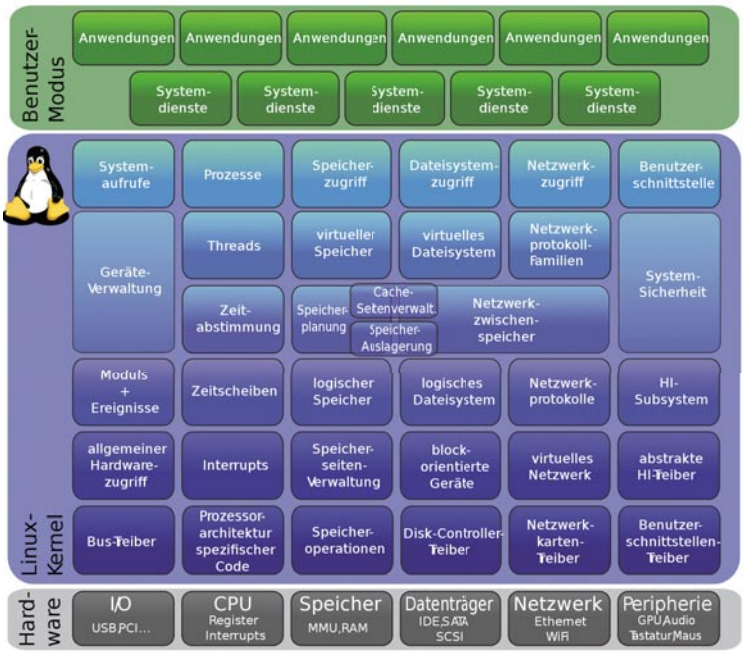


Bild 1: Der Linux-Kernel mit seinen Hardware-unabhängigen Modulen wird umrahmt von den GNU-Komponenten Benutzerschicht und Hardware-Anpassung. (Quelle: [15])

abfinden. Stallman rief deshalb 1983 das GNU-Projekt (GNU – **G**NU is **n**ot **U**NIX), ein offenes Pendant zu UNIX, ins Leben. Dessen Quellcode lag offen, alle Entwickler waren aufgerufen, an der Software mitzuarbeiten, nur UNIX-kompatibel musste sie sein. So entstand nach und nach ein System mit vielen kompatiblen Bausteinen. Nur das Wesentliche fehlte – der sogenannte Kernel, ohne den kein System läuft. Erst das 1987 erschienene Minix-System enthielt einen sogenannten Mikro-Kernel als UNIX-Clone, mit dem sich die GNU-Komponenten auf einem normalen PC (UNIX ist ja ein Workstation-basiertes System) nutzen ließen. Minix wurde zwar mit Quellcode, aber dennoch unter proprietärer Lizenz verkauft. Erst 1991 schrieb der damalige Student Linus Torvalds das an Minix orientierte Linux. Es entstand quasi als Minix-kompatibles Betriebssystem nebenbei aus einer Terminal-Emulation, die der finnische Student aus Lerninteresse schrieb. Der Name Linux (ein Namensderivat aus Linus und Minix) ist einem Administrator des Uni-Netzwerks zu verdanken, er vergab diesen Namen für Torvalds als Account. Um das System durch die Mithilfe vieler Entwickler weiterentwickeln zu können, stellte Torvalds es wenig später unter die GNU-GPL (GPL – **G**eneral **P**ublic **L**icense – freie, nicht mit Lizenzgebühren belegte Software, die von jedermann inklusive Quellcode weitergegeben, weiterentwickelt, verändert oder veröffentlicht werden darf).
Damit gab es nun ein freies, UNIX-kompatibles Betriebssystem, um das sich, als Linux-Kernel, die GNU-Systembestandteile anordnen – eine perfekte Plattform für Software-Entwickler!

Die Linux-Struktur

Abbildung 1 zeigt die Kernel-Struktur von Linux. Der Kernel selbst liegt im Quellcode vor, der sich grob unterteilt in den plattformunabhängigen Betriebssystemkern (Speicherverwaltung, Multitasking etc.), die auch weitgehend platt-

formunabhängigen Gerätetreiber und den prozessorspezifischen Code. Auf Letzterem bauen die beiden anderen Teile auf. Durch diese Strukturierung läuft Linux auf vielen unterschiedlichen Prozessoren und unterstützt sehr viele unterschiedliche Geräte.
Im Embedded-Bereich wird der Kernel aus dem Quellcode genau für die vorhandene Hardware (Prozessor und Geräte) kompiliert, dadurch vermeidet man für diese Konfiguration nicht benötigte Kernel-Bestandteile und der so erzeugte Kernel wird „schlank". Der für unser System umfasst weniger als 2 MByte.
Die verwendeten Teile werden im Entwicklungssystem in einer grafischen Oberfläche ausgewählt, Gerätetreiber lassen sich monolithisch in den Kernel einkompilieren oder zur Laufzeit als Module nachladen. Diese spielen bei Linux auf dem PC eine große Rolle, weil die Linux-Distributionen für jede erdenkliche Hardware Treibermodule mitliefern. Das kommt natürlich für Embedded-Linux-Systeme mit ihrem punktgenau auf die verwendete Hardware angepassten Kernel aus Ressourcen-Gründen nicht in Frage, weshalb hier die Module eher während der Entwicklung wichtig sind, weil sie sich zur Laufzeit neu laden lassen. Positiver Effekt: Bei Änderungen ist kein Reboot des Systems nötig.
Das Ergebnis dieser Arbeit: Bei den Anwendungen merkt man von der verwendeten Hardware kaum etwas, das System „verhält" sich neutral.
Auf den Kernel setzen die schon zur Anwendungsschicht zählenden Systembibliotheken auf (libc, Abbildung 2). Während auf PCs fast nur die GNU-libc (glibc) zum Einsatz kommt, greift man für die kleinen Embedded-Systeme zur deutlich kompakteren und hierauf optimierten uClibc [3].Sie ist zu glibc kompatibel, bei eingeschränktem Funktionsumfang, und hochgradig konfigurierbar: Teile, die nicht verwendet werden, müssen auch nicht kompiliert werden.
Für den tieferen Einstieg in den Kernel und die Softwareentwicklung verweisen wir auf die kommenden Folgen sowie auf das Muss an Internetseiten zu diesem Thema: [4], [5], [6] und [7].

Linux-Anwendungen und Entwicklungs-Tools

Heute gibt es so viele Linux-Anwenderprogramme, dass der Nutzung von Linux sowohl im Desktop-Bereich, wie z. B. Büroanwendungen, Internet, Spiele, Multimedia-Anwendungen, Grafik, CAD usw., als auch für spezielle Anwendungen, z. B. in der Steuerungstechnik, oder auf portablen Geräten nichts mehr entgegensteht. Aber auch der Programmierer, an den wir uns ja mit der LCU 1 wenden, hat eine große, freie Vielfalt an Werkzeugen zur Lösung seiner Aufgabe zur Auswahl wie beispielsweise:

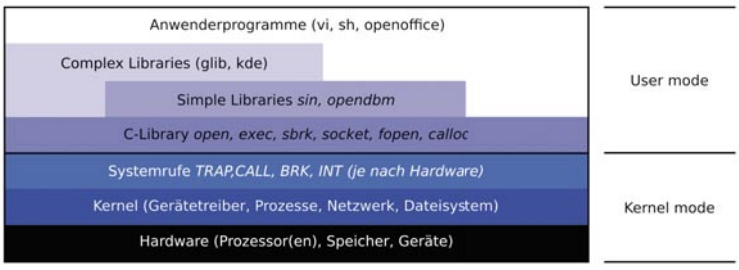


Bild 2: Das GNU-Schichtenmodell (Quelle: [16])

- Laufzeitbibliotheken für C und C++ (GNU-Libraries)
- Qt/Embedded für die Entwicklung grafischer Anwendungen auf verschiedenen Hardware-Plattformen, speziell auf mobilen Geräten mit begrenzten Speicherkapazitäten
- den freien, modularen Web-Server „Lighttpd" mit CGI (CGI ist eine Schnittstelle zwischen Web-Server und externer Software)

Diese Software-Bausteine machen die Entwicklung von grafischen HTML-/CGI-Anwendungen besonders einfach, auch der Einsatz eines solchen Embedded-Systems wie dem LCU 1 als Bedienterminal für Serverapplikationen ist so einfach lösbar.
Für Embedded-Systeme ist die Bedienung per Kommandozeile sehr wichtig. Hier gibt es ein spezielles Kommandozeilenprogramm, „busybox" [8], das alle wichtigen Tools und Kommandos hierzu in einem einzigen Programm vereint. Abgesehen von den verwendeten Systembibliotheken ist busybox bei unserer LCU 1 nur ca. 600 KByte groß.
Als Compiler kommt der GNU-Compiler (gcc) zum Einsatz. Für die grafische Interaktion mit dem Anwender kommt die Bibliothek „directfb" [9] zum Einsatz. Sie ist schlank und setzt direkt auf das Linux-Framebuffer-Device auf. Das stellt einen Speicherbereich zur Verfügung, über den direkt die Displaypixel manipulierbar sind, das Display ist also direkt als Speicherbereich in die Anwendungsschicht eingeblendet. „directfb" bietet, darauf aufsetzend, die Möglichkeit, Grafiken, z. B. PNG, anzuzeigen, Rechtecke und Linien zu zeichnen, Text auszugeben oder auf Ereignisse (Maus, Touchscreen) zu reagieren. Auf der Basis von „directfb" kann man in C eigene grafische Oberflächen erstellen.
Alle Teile eines Embedded-Linux-Systems (u. a. Compiler, Bootloader, Systembibliotheken, Kernel, Anwendungen, evtl. grafische Bibliotheken) von den richtigen Stellen in den richtigen Versionen herunterzuladen und für das konkrete System anzupassen, ist sehr aufwändig. Diese Arbeit wird sehr erleichtert von „buildroot" [10]. Dies ist eine Sammlung von Skripten und Makefiles, die vom Herunterladen über das Kompilieren bis zum Erstellen eines Dateisystems alle nötigen Schritte übernimmt. buildroot ist hoch flexibel und wird grafisch konfiguriert. Auch deshalb werden wir in der Folge eine auf buildroot basierende Entwicklungsumgebung verwenden, die sich durch Konfiguration von buildroot oder Hinzufügen eigener Programme erweitern lässt.

Bootloader „U-Boot"

Der Bootloader ist verantwortlich dafür, dass nach dem Start des Systems der Linux-Kernel ordnungsgemäß in das System geladen und gestartet wird. Er ist Hardware-abhängig und sehr flexibel an die eigene Hardware-Plattform anpassbar. „U-Boot" ist ein sehr kompakter Standard-Bootloader für Embedded-Systeme, der den Vorteil hat, auch vom NOR-Flash, NAND-Flash, Dataflash, USB, Netzwerk etc. booten zu können. Insbesondere das Booten per Netzwerk ist für Embedded-Systeme sehr nützlich, dadurch ist keine Neuprogrammierung des betreffenden Gerätes nötig. Die von uns in der LCU 1 verwendete ARM-Version benötigt nur 200 KByte Speicher und ist damit besonders kompakt ausgeführt.
In [11] und [12] ist der Bootloader ausführlich beschrieben, wir werden auch im Verlauf der Serie noch darauf zurückkommen.

Schaltungsbeschreibung des LCU 1

Auf die detaillierte Beschreibung aller Schaltungsteile wird aufgrund des großen Umfangs in diesem Artikel bewusst verzichtet. Wir beschränken uns deshalb auf die Beschreibung des in Abbildung 3 gezeigten Blockschaltbildes, für den Nutzer des fertig aufgebaut vertriebenen Boards sicher wesentlich interessanter, weil alle Möglichkeiten auf einen Blick zu sehen sind. Zentrales Element ist der Atmel-ARM-Prozessor AT91SAM9261, der mit 190 MHz getaktet wird.
Als externer Speicher sind auf dem Board 64 MByte SDRAM, 256 MByte NAND-Flash sowie per SPI 8 MByte Data-Flash vorhanden.
Als Bedien-Interface kommt ein 14,5-cm-TFT-Touchpanel zum Einsatz, das eine Auflösung von 320 x 240 Pixel aufweist. Es wird direkt vom LCD-Controller über die digitale 18-Bit-RGB-Schnittstelle des Prozessors angesteuert, ein Touch-Controller wertet die Befehle des Touchscreens aus und schickt sie per SPI-Schnittstelle an den Prozessor.
Für die Anbindung von externen USB-Geräten stehen zwei USB-Host-Schnittstellen zur Verfügung, dazu eine USB-Device-Schnittstelle für die Anbindung z. B. an einen PC. Für die Anbindung an ein Ethernet-Netzwerk ist über den bekannten Ethernet-Transceiver DM 9000 (siehe auch Artikel „IP-SL232" in dieser Ausgabe) eine 10/100-Mbit-Ethernet-Schnittstelle realisiert. Über eine 2-Draht-USART-Schnittstelle ist ein Debug-Interface für die Kommunikation beim Debuggen/Programmieren vorhanden. Über insgesamt 10 IO-Ports sind 4 Relais-Schaltausgänge mit potentialfreien Kontakten sowie 4 digitale und 2 analoge Signal-Eingänge zugänglich.
Natürlich fehlen auch serielle Schnittstellen nicht, jeweils ein UART-Port bedient ein Terminal für eine RS485-Schnittstelle sowie einen 9-poligen D-Sub-Port für EIA/RS232. Auch eine Tonausgabe über einen Signalgeber fehlt ebenso wenig wie 2 Bedientaster, ein Schacht für microSD-Karten und ein zentraler Reset-Taster.
Die Spannungsversorgung erfordert lediglich eine Eingangs-Gleichspannung von 10 bis 30 V, alle anderen Spannungen werden über Spannungswandler auf dem Board erzeugt.

Zweiter Teil: erste Entwicklungsschritte

Im nächsten Teil der Serie zeigen wir den Weg zur Erstellung eigener Software für die Nutzung auf der LCU 1 in Form der Entwicklungsumgebung auf.

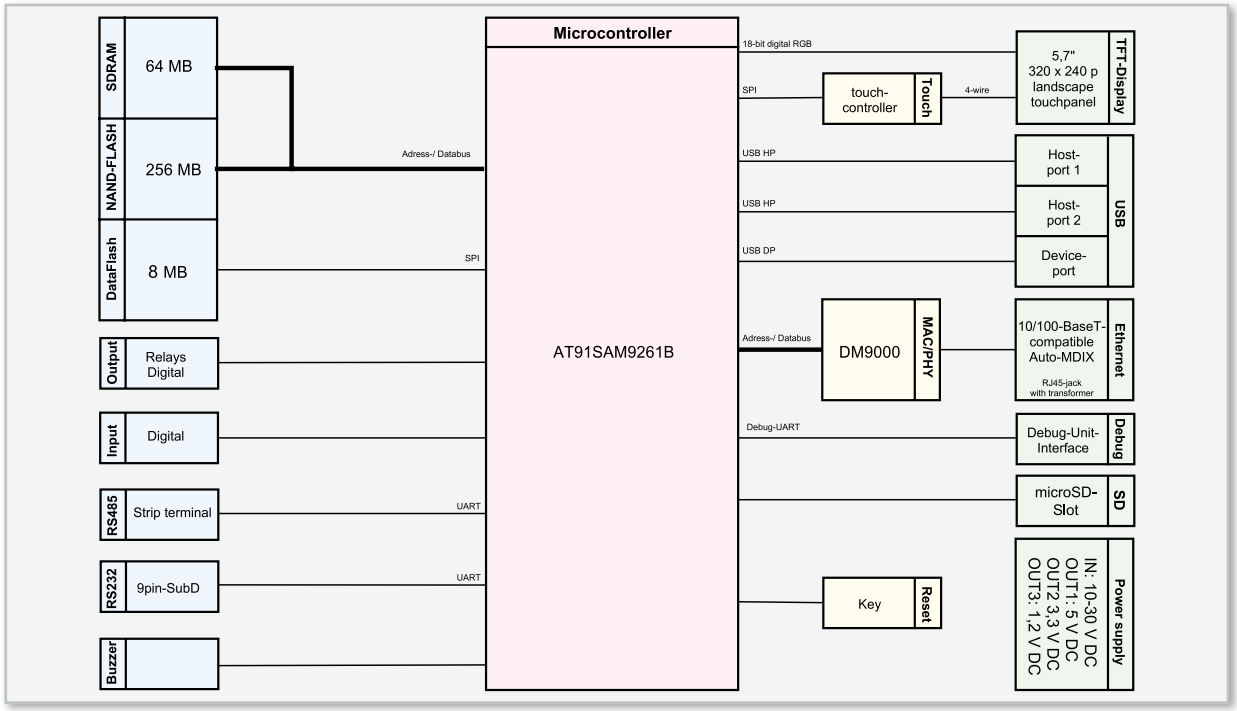


Bild 3: Blockschaltbild des LCU 1

Links:

[1] http://infocenter.arm.com
[2] www.atmel.com/at91
[3] www.uclibc.org
[4] www.kernel.org
[5] www.kernelplanet.org
[6] www.kernelnewbies.org
[7] www.ibm.com/developerworks/linux/library/l-linux-kernel
[8] www.busybox.net
[9] www.directfb.org
[10] http://buildroot.uclibc.org/
[11] www.linux-arm.org/LinuxBootLoader/WebHome
[12] www.denx.de/wiki/U-Boot/WebHome
[13] www.at91.com
[14] www.at91.com/linux4sam

Quellen:

[15] http://upload.wikimedia.org/wikipedia/commons/f/f8/Linux_Kernel_Stuktur.svg
[16] http://upload.wikimedia.org/wikipedia/de/b/b3/Linux_Schichten.svg