LINUTRONIX

Fastboot

Wir haben die Lösung!

- Ein Linux, das „sofort“ nutzbar ist.
- Mit graphischer Bedienoberfläche.
- Mit aktiven Schnittstellen für Netz, Feldbusse, ...

Bootloader

Deutlich mehr Zeit lässt sich beim Bootloader einsparen und beim Laden/Kopieren und Entpacken des Kernels. Auch hat die Konfiguration des Kernels Auswirkungen auf die Bootzeit, ebenso wie das Linken des Applikationsprogrammes und die Kompressionsart des Kernels.

Ersetzt man den Bootloader (Beispiel: U-Boot) durch einen **IPL** (Initial Program Loader), also zum Beispiel durch einen funktional aufgewerteten Bootstrap Loader, der selber den Linux Kernel laden kann, dann wird der gesamte Vorgang essentiell beschleunigt.

Bootzeit Kernel

Die Bootzeit des Kernels selbst wird hauptsächlich durch folgende Punkte beeinflusst:

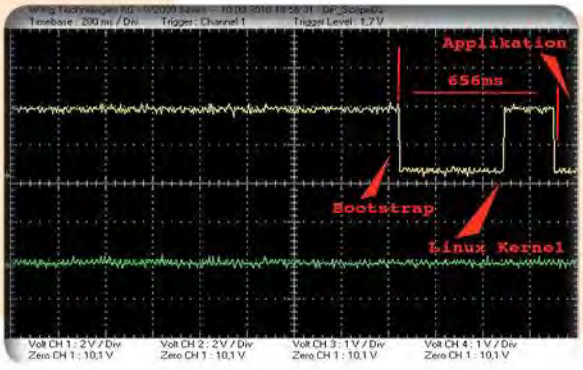
- Konfiguration und Build
- Kompression
- Bootparameter
- Treiberinitialisierungen
- Rootfilesystem

Von den oben genannten Parametern sei besonders auf das Rootfilesystem bzw. das dafür verwendete Dateisystem hingewiesen. Ein aktuell noch sehr häufig verwendetes Flashfilesystem ist das jffs2. Dieses hat aber einige Nachteile, vor allem benötigt es umso mehr Zeit zum Einbinden, je größer das Flash ist. Diese Nachteile werden bei einem Flashfilesystem wie **UBIFS** vermieden. Für Flash basierte Speicherkarten als Bootmedium sind wieder andere Filesysteme zu bevorzugen.

Bootzeit Applikation

Hier kann sehr viel Zeit eingespart werden, zum Beispiel durch ein Optimieren des Linkens. Eine weitere wertvolle Hilfe liefert die Analyse des Initprozesses. Das Tool „bootchart" bietet hierfür in graphischer Form wertvolle Unterstützung.

Die Informationen aus dem Diagramm liefern auf einfache und anschauliche Art Hinweise, wo im Bootprozess Zeit eingespart werden kann. Bei einem Projekt auf Basis eines Atmel AT91 Prozessors, der mit dem mitgeliefertem Board Support Package (inkl. busy box) ca. 11 Sekunden benötigte, um eine Benutzeranwendung zu aktivieren, gelang es uns mit den zuvor aufgezählten Maßnahmen, die Bootzeit auf ca. 600 msek. (!) zu verkürzen.

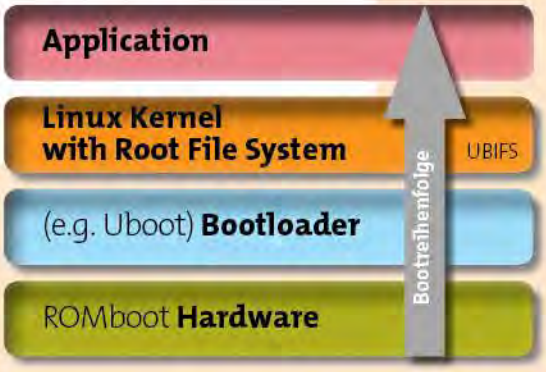


Bootzeit << 1 sec

Embedded Systeme haben Ihre eigenen Anforderungen. Einschalten und Benutzen können, das ist der Wunsch vieler Anwender.

Genau betrachtet geht es nicht nur ums Booten, es geht darum, dem Benutzer in möglichst kurzer Zeit ein benutzbares System zu offerieren. Ein System, das bereits aktuelle Informationen anzeigt oder, besser noch, bereits aktiv mit seiner Umwelt kommuniziert.

Bis dies möglich ist, sind allerdings viele Dinge abzuarbeiten. Dazu müssen verschiedene Software Programme Hand in Hand zusammen arbeiten. Es beginnt mit dem Urlader, den sog. Bootstrap Loader, der den Bootloader startet. Dieser wiederum lädt den Kernel, kopiert ihn in den Arbeitsspeicher und entpackt ihn dann. Nach der Initialisierung der restlichen Hardware durch den Kernel setzt dieser das System auf und startet den User Prozess. Das untenstehende Bild visualisiert diesen Ablauf.



Hardware Optimierung

Bevor aber die Software zum Zuge kommt, ist das System erst einmal unter Strom zu setzen. Wenn hier keine optimierte Hardware eingesetzt wird, kann man Bootzeiten unter 1 Sekunde sofort vergessen. An dieser Stelle sind die Hardware Designer gefragt, die folgende Fragen beantworten müssen:

- Wieviel Zeit wird benötigt, bis eine stabile Spannung vorhanden ist und der Controller den Reset-Modus verläßt?
- Resetlogik und ihre Auswirkungen auf die Bootzeit.
- Bootlogik - von wo wird gebootet? Gibt es eine interne Logik hierfür? Was ist das optimale Bootmedium. Wie ist es anzubinden? Busbreite? Zugriffszeiten?

Die Zeitspanne vom Anlegen der Versorgungsspannung bis zum Starten des Bootloaders ist im Allgemeinen vernachlässigbar (sofern das Hardware Design die oben beschriebenen Fragen richtig gelöst hat).

CHRISKNEIFEL

Suspend Modi und weitere Möglichkeiten

Manchmal ist es nicht möglich, den Bootvorgang so stark wie in dem genannten Beispiel zu verkürzen oder es sind andere Anforderungen vorhanden, die diesen Ansatz zum schnellen Booten nicht zulassen. Vielleicht muss das System aktiv auf ein externes Signal lauschen und beim Eintreffen des Signals „schlagartig" wach werden. Oder das System muss aktiv an seinen I/O Schnittstellen „lauschen", ob eine Service-Applikation andockt, die einen Update durchführen möchte. Was nun das schnelle Booten in diesen Fällen angeht, so gibt es auch hierfür bewährte Lösungsansätze. So kann es für Systeme, die auf ein „WakeUp" Signal warten, sinnvoll sein, diese in einen Suspend-Modus zu setzen. Dies kann ein vom CPU Hersteller vorgesehener Suspend-Modus der CPU alleine sein oder, falls das System-Design dies unterstützt, ein Suspend-Modus nahezu des gesamten Systems.

Bei sehr komplexen Systemen mit vielen Prozessen ist die kritische Phase oft das Starten der vielen User-Applikationen. Hier können Ansätze wie „systemd" oder „readahead" zu deutlichen Verbesserungen der Bootzeit führen. Eine weitere Möglichkeit besteht darin, sogenannte „Schnappschüsse" des Systems während der Startphase und/oder nach dem Starten des ersten Benutzerprogramms zu erstellen. Ausgehend von den dort gewonnenen Informationen wird der Startvorgang optimiert, d.h. es wird eine spezielle Art des „suspend to xx" umgesetzt.