Entwicklung eines Linux Gerätetreibers am Beispiel eines e-Paper Displays

Anna-Lena Marx

11. Januar 2016

Abstract

Inhaltsverzeichnis

1 Einleitung

Im gerade aufstrebenden Gebiet der eingebetteten Systeme spielt Linux nicht nur als schlankes, gut portierbares Betriebssystem eine tragende Rolle. Auch die Möglichkeit neue Hardware, beispielsweise Sensorik einfach in bestehende Systeme integrieren zu können macht Linux zu einer interessanten Plattform. Während in es in vielen Fällen ausreicht solche Hardware über sehr gut dokumentierte und einfach zu verwendende Schnittstellen im `Userspace` anzusprechen, gibt es doch Anwendungsfälle die Hardwaretreiber im `Kernelspace` erfordern.

cZu genannten Fällen zählen beispielsweise zeitkritische Treiber, die darauf angewiesen sind priorisiert und zu festen Zeitpunkten abgearbeitet zu werden, oder auch Hardwaretreiber im Android-Umfeld, die schon aufgrund des Designs des Systems beziehungsweise der Rechteverwaltung nicht im Userspace laufen dürfen.

Diese Projektarbeit soll sich mit solch einem Kerneltreiber auf einem eingebettetem System beschäftigen und exemplarisch darstellen, wie ein Linux-Treiber aufgebaut ist und funktioniert. Dazu wird an einem Beaglebone Black, einem eingebetteten Entwicklerboard, über die `UART`-Schnittstelle ein e-Paper-Display durch ein Linux-Kernel-Modul angebunden. Das Modul soll dem Nutzer die linuxtypischen Treiberschnittstellen bereitstellen, um die Kommunikation mit der Hardware zu ermöglichen.

Der Fokus soll hierbei vor allem auf allgemeinen Prinzipien und Funktionsweisen von Linux-Treibern liegen und erklären was in diesem komplexen, wenig bekannten Bereich ablaufen muss um Hardware so selbstverständlich verwenden zu können, wie dies in heutigen Linux-Systemen der Fall ist.

2 Hardware

2.1 Beaglebone Black

Zur Durchführung der Projektarbeit wird das Beaglebone Black, ein eingebettetes Entwicklerboard auf Basis des ARM-Prozessors AM335x von Texas Instruments eingesetzt. Das Beaglebone bietet mit einer Taktrate von 1GHz (Singlecore), 512MB RAM und vielen zugänglichen Hardware- und Debug-Schnittstellen, sowie einer sehr guten Dokumentation ideale Voraussetzungen zur Hardwareintegration.

2.2 Waveshare e-Paper-Display

Bei dem e-Paper-Display handelt es sich um ein Modell des Herstellers Waveshare mit einer Auflösung von 800 x 600 Pixeln und einer Größe von 4,3 inch, dass sehr einfach über die bekannte `UART`-Schnittstelle angesprochen werden kann.

3 Aufbau und Vorbereitungen

Das Beaglebone wird mit Ubuntu-Linux und dem Kernel betrieben. Dabei ist der Treibercode nicht von einer Linux-Distribution abhängig.

Zuerst wurde für diese Arbeit zu Beginn ein Arch-Linux System mit einem Mainline-Kernel der Version ... benutzt, da dieses Setup größtmögliche Freiheit in der Konfiguration des Systems und der Verwendung von Custom-Kernels bot. Leider gab es in dieser Konfiguration ein Problem bei der Verwendung der benötigten `UART`-Schnittstellen dessen Lösung den Rahmen dieser Arbeit sprengen würde und einen Wechsel unvermeidlich werden lies.

Der Treiber wird für die Kernelversionen 4.x der ARM-Plattform geschrieben und ist, solange es keine größeren Änderungen der verwendeten APIs gibt, für jeden entsprechenden Kernel kompilierbar.

3.1 Aufbau der Hardware

Das e-Paper Display besitzt ein `UART`-Interface, welches die Kommunikation zwischen Beaglebone und Display über zwei Pins ermöglicht. Dazu wird die `DIN`-Leitung des Displays an den `TXD`-Pin des `UART-1`-Interface des Beaglebones angeschlossen. Ebenso wird mit der `DOUT`-Leitung und dem `RXD`-Pin des gleichen Interfaces verfahren. Die Leitungen `RST`, der Reset und `WAKEUP` des Displays werden an den `GPIO`[1]-Schnittstellen des Beaglebone angelegt und sorgen dafür, dass der Displayinhalt gelöscht, bzw. das Display aus einem Ruhezustand geholt werden kann. Zuletzt müssen noch Versorgungsspannung (`VCC`, 5V) und Erdung (`GND`) an die entsprechenden Pins des Beaglebone angeschlossen werden.

Das Beaglebone selbst wird über ein USB-Kabel an den Hostrechner angeschlossen und kann darüber über das `ssh`-Protokoll erreicht werden. Um die zusätzliche Hardware versorgen zu können, muss das Board zusätzlich über ein 5V-Netzteil mit Strom versorgt werden. Um schon ab dem Bootvorgang Kernellogs zeitgleich lesen zu können wird die serielle Debugging-Schnittstelle des Beaglebone mithilfe eines USB-Serial-Wandlers und dem Programm `Minicom` ausgelesen.

3.2 Vorbereitungen zum Coding

Kernelmodule müssen genau zu der Kernelversion, also den Schnittstellen genau des Kernels passen auf dem sie ausgeführt werden sollen. Daher müssen die Quelltexte des

[1] Fußnote zu GPIO schreiben

Kernels vorliegen um dafür ein Kernelmodul zu erstellen. Sind Zielsystem des Treibers und dass auf dem das Modul kompiliert werden soll identisch, Handelt es sich wie im Fall dieser Arbeit um unterschiedliche Plattformen und Kernelversionen, muss der Quelltext der genau passenden Kernelversion, sowie ein Compiler für die Zielplattform geladen werden. Natürlich kann auch direkt auf der Zielplattform, dem Beaglebone entwickelt werden, allerdings aufgrund dessen geringer Leistungsfähigkeit nicht empfehlenswert.

Für die hier verwendete Kernelversion können die Quelltexte wie folgende geladen und kompiliert werden. `CROSS_COMPILE` gibt dabei an, welcher (Cross)-Compiler verwendet werden soll.

Listing 1: Laden der Kernelquellen

List of Listings

Abbildungsverzeichnis