Erste Erfahrungen mit Gnublin

Hubert Högl, 2012-06-18, EPJ-13

Dieser Text gibt einige Erfahrungen wieder, die ich als Veranstalter eines Embedded Linux Kurses seit Mitte März mit dem Gnublin Board gemacht habe. Am Schluss beschreibe ich kurz die Versuche, die wir mit dem Gnublin Board gemacht haben.

Die Hardware

Die Gnublin 1.3 Platine funktioniert meist problemlos. Die Grösse und der Funktionsumfang sind gut für eine Vielzahl von Experimenten geeignet. Vor allem die Erweiterbarkeit über die USB OTG Schnittstelle hat sich als sehr praktisch herausgestellt, so dass man gezielt die für ein Experiment notwendigen Peripheriebausteine anschliessen kann, z.B. WLAN, Bluetooth, eine Sound Codec oder ein GPS Modul. Auch die MicroSD Karte als Speicherort für Bootloader, Kernel und Root Filesystem ist eine einfache und praktische Lösung. Durch den geringen Stromverbrauch von unter 100 mA eignet sich das Board auch gut für batteriebetriebene Geräte.

Es gibt trotzdem ein paar Verbesserungspunkte:

- Man wünscht sich gelegentlich an den Schraubanschlüssen eine 5V Spannung für Erweiterungen. Findige Leute haben sich deshalb eine "Spezial-Jumper" gebaut, der alle drei Pins des "Power-Supply" Pfostenleiste überbrückt.
- Mit den 8 MByte Hauptspeicher kommt man zwar erstaunlich weit, manche Aufgaben sind aber deswegen nicht möglich. Ein paar Beispiele sind: (1) Der GNU C Compiler läuft mit 8 MByte leider nicht. (2) Manche Linux Distributionen legen am Anfang ein paar Megabyte temporäre Dateien als "tempfs" ab, das ist bei 8 MByte nicht möglich. (3) Manche einfach erscheinenden Dienstprogramme, z.B. "apt-get" brauchen so viel Speicher, dass sie in 8 MByte nicht ausführbar sind. (4) Programme aus dem Multimedia-Bereich, z.B. ogg123 oder mpg123 brauchen zum Entkomprimieren viel Hauptspeicher, so dass die erzielbare Datenrate bei 8 MByte deutlich sinkt. (5) Grössere Interpreter wie Python passen nur knapp in den freien Speicher, dadurch braucht der Start dieser Interpreter inakzeptabel viel Zeit (5 bis 10 Sekunden).

  32 MByte Speicher sind aber bereits in Sicht (Version 1.5). Damit sollten die fünf genannten Beispiele dann kein Problem mehr sein.
- Fast alle neuen Anwender machen die Erfahrung, dass nach der ersten Sitzung mit dem Board verstärkt Fehler beim Booten auftreten. Das liegt meist daran, dass das "ext2" Filesystem auf der MicroSD Karte plötzlich Fehler hat, die durch spontanes Abbrechen des Linux Kernels verursacht werden. Seit jeher wird Linux geordnet mit Kommandos wie "shutdown" oder "halt" heruntergefahren, nicht einfach durch Ziehen des Steckers abrupt beendet. Hier ist meist der Punkt, an dem man sich mit der Installation von Linux auf dem Entwicklungsrechner anfreunden muss, nur damit schafft man es, die Karte in einem "file system check" (fsck) wieder zu reparieren. Eine Verbesserung wäre es, die Karte mit dem "ext3" Filesystem auszustatten, das auf Fehler gutmütiger reagiert. Aber auch damit kann ein fsck nötig sein.

  Bei wenigen Teilnehmern war die MicroSD Karte gleich am Anfang durch unkontrollierte Änderungen (wahrscheinlich durch wildes Herumklicken) in einem Zustand, dass der Inhalt komplett wiederhergestellt werden musste. Das ist mit dem grafischen "gnublin-installer" relativ einfach. Es ist ausserdem nützlich, wenn man sich gelegentlich ein "Image" der Karte macht, z.B. mit dem Werkzeug "dd", vor allem, wenn man Änderungen an der Karte vorgenommen hat. Dieses Image kann man bei einem späteren "Crash" zur Rekonstruktion verwenden.
- Eine zweite serielle Schnittstelle (UART) wäre an der Pfostenfeldleiste für Erweiterungen manchmal wünschenswert. Der LPC3131 hat aber leider nur einen UART, der schon für die Konsole verwendet wird.

Der Kernel

Die Kernel Version

Gnublin verwendet den LPC3131 Mikrocontroller von NXP, das ist auch der Grund, warum wir (leider) ziemlich stark auf den Kernel mit der Version 2.6.33 festgelegt sind. Mitarbeiter von NXP haben damals (2010) bei der Anpassung diesen Kernel verwendet und leider bis jetzt keinen Update mehr für eine neuere Kernel-Version gemacht. Für vieles was man mit Gnublin machen kann stört das überhaupt nicht, aber gelegentlich wäre ein neuerer Kernel ganz praktisch, zum Beispiel wenn man

- neue Peripheriegeräte anschliessen möchte, die es zur Zeit der Version 2.6.33 noch nicht gab. Ein Beispiel: Vor ein paar Tagen habe ich mir einen D-Link Go DWA-123 WiFi Adapter gekauft (USB), der den Treiber rt2800usb verwendet. Diesen Treiber gibt es zwar im Gnublin Kernel, allerdings kennt sie noch nicht den RT3370 Chip, der in diesem Adapter verwendet wird. Aus diesem Grund kann man beim Gnublin Kernel tendenziell nur ein wenig "reifere" Peripheriegeräte verwenden, z.B. die Asus WL-167G WiFi Adapter V1 und V2. Damit die Version V3 dieses Adapters funktioniert, musste aber bereits aus dem 2.6.39 Kernel der Treiber staging/rtl8712/ in den Gnublin Kernel 2.6.33 eingebaut werden. Es ist sicher wünschenswert, dass man diese Vorgehensweise nur ganz selten anwendet.
- aus der Weiterentwicklung des Kernels allgemein Nutzen ziehen möchte. Die meisten Neuerungen sind sicher für Gnublin nicht von Bedeutung, allerdings gibt es gelegentlich Verbesserungen in den Frameworks für die Peripherie, z.B. GPIO, PWM und I2C.

Mögliche Auswege aus der Festlegung auf 2.6.33 wären, sich die Mühe zu machen, selber die Anpassung für LPC3131 auf einen neueren Kernel 3.x zu machen (das erfordert aber schon sehr fortgeschrittene Kenntnisse), oder gleich auf einen anderen Controller umzusteigen, der in Linux "Mainline" unterstützt wird. Ein möglicher Kandidat wäre der LPC3250.

Den Kernel kompilieren

Mit ein wenig Anleitung schaffen es die Kursteilnehmer relativ schnell, einen neuen Kernel und die dazugehörigen Module zu kompilieren. Auch die Auswahl der gewünschten (einfachen) Kernel-Features durch "make menuconfig" ist mit ein wenig Übung zu bewältigen. Hier ist vor allem die hohe Zahl und die tiefe Verschachtelung der Menüeinträge am Anfang schwierig. Die Frustration steigt auch durch die vielen Einträge, deren Bedeutung sich nur dem professionellen Kernel-Entwickler erschliessen.

Beim Umgang mit dem Kernel merkt man schnell, dass man sich auch um die *Organisation* von Quelltexten und Daten kümmern muss. Es fängt an mit dem riesigen Quelltextbaum des Kernels, an dessen oberster Stelle auch die Konfigurationsdatei .config sitzt. Nach erfolgreicher Konfiguration/Kompilierung erhält man als Ergebnis einen neuen Kernel arch/arm/boot/zImage und einen Satz an Modulen, die man mit make modules_install INSTALL_MOD_PATH=... an einen geeigneten Ort zusammen mit dem Kernel und der aktuellen Konfigurationsdatei auf dem Entwicklungsrechner ablegt. Von hier muessen die Kerneldateien auf die MicroSD Karte übertragen werden, so dass sie von Gnublin beim nächsten Booten verwendet werden.

Wenn man nun mit mehreren Kernelkonfigurationen Experimente macht und vielleicht sogar auch am Quelltext Änderungen macht, muss man sich bereits einen ordnenden Plan machen, um nicht die Übersicht zu verlieren, welche Features in welcher Version verwendet werden. Auch kann es passieren, dass ein neu komplierter Kernel nicht bootet, so dass man wieder auf den letzten funktionierenden Kernel zurückgehen muss.

Der Anfänger kann sich an diesem Punkt schnell verheddern, so dass nur ein kompletter Neuanfang wieder Ordnung und Übersicht bringt. Hilfreich wäre ein einfaches Kommandozeilenwerkzeug, das den Anfänger bei der Erstellung eines neuen Kernels stärker führt und die Ergebnisse reproduzierbar speichert.

Stark verbesserungsfähig ist die Art und Weise, wie der Kernelcode zur Zeit auf gnublin.org verwaltet wird. Es ist momentan eine Mischung aus Git Repository und klassischen Patches, bei deren Anwendung

auch viel schiefgehen kann. Als provisorische Lösung habe ich ein eigenes git Repository mit dem Branch "gnublin" für den Gnublin Kernel eingerichtet, siehe

http://elk.informatik.fh-augsburg.de/hhwiki/EmbeddedLinux2012

Nachdem man Änderungen am Kernel gemacht hat, stellt sich schnell die Frage, ob alle gewünschten Kernel-Features noch funktionieren, auch die, bei denen man gar keine Änderungen gemacht hat. Eigentlich müsste man nun in der Gnublin Konsole viele Punkte überprüfen, z.B. ob die gewünschten Netzwerktreiber vorhanden sind, ob das sysfs vorhanden ist, ob sich die PWM und ADC Treiber laden lassen, ob ext3 im Kernel aktiviert ist und so weiter. Hier wäre ein kleines Testsystem von Vorteil, das Konsoleneingaben automatisiert ablaufen lassen kann, z.B. mit dem Werkzeug "expect".

Das Root Filesystem

Das momentan mit Gnublin ausgelieferte Root Filesystem basiert auf dem ELDK 5.0 (siehe http://www.denx.de). Mittlerweile gibt es die Version 5.2, das zwei für Gnublin schöne Verbesserungen gegenüber 5.0 hat:

Es gibt kleinere Varianten, die nur noch die wirklich gebrauchten Programme dabei haben, zum Beispiel brauchen wir gar keine grafischen X-Window Programme. Somit schrumpft das Root Filesystem von derzeit über 600 MByte (5.0) auf 100 bis 200 MByte. Das wirkt sich deutlich aus auf alle Operationen, die mit Download, Kopieren, Sichern und Beschreiben des Root Filesystems zu tun haben.

Die zweite Verbesserung betrifft das Paketmanagement. Die Version 5.2 kommt mit dem ipkg Paketmangager, der Binärpakete aus dem Netz holen und installieren kann. Ein eigenes Paketrepository wäre für Gnublin ideal, da sich fast jeder, der intensiv mit dem Board arbeitet, eigene Zusätze für sein Root Filesystem zusammenbaut, diese aber oft nur auf der lokalen MicroSD Karte gespeichert werden. Diese Zusätze kommen oft aus dem Debian/ARM ("armel") Projekt, man kann die vorkompilierten Programme direkt bei Gnublin übernehmen, oder sie kommen durch selber crosskompilierte Programme. Die Reproduzierbarkeit von Änderungen am Root Filesystem ist also, ähnlich wie schon im Abschnitt über den Kernel angesprochen, mangehaft. Die Anpassung von ELDK 5.2 für Gnublin wird also immer dringender.

Alternativ gibt es mittlerweile ein für Gnublin angepassstes Debian 6 (siehe Projekte weiter unten). Damit hat man dann den "dpkg" Paketmanager, der allerdings nur einzelne .deb Pakete installieren kann. Das Holen aus dem Netz und die Auflösung von Abhängigkeiten muss man manuell erledigen. Mit den "apt" Utilities könnte man das zwar komfortabel erledigen, sie laufen aber leider nicht mit 8 MByte Speicher (mit 32 MByte vielleicht schon).

Arbeiten auf dem Entwicklungsrechner

Ein deutlicher Wohlfühlfaktor im Umgang mit dem Gnublin Board ist die Fähigkeit, sich auf dem Entwicklungsrechner unter Linux zurechtzufinden. Einige Aufgaben, für die man diesen Rechner verwendet, sind:

1. Betrachten der Gnublin Konsole (Terminalemulation mit picocom o.ä.)
2. Austausch von Dateien mit Gnublin (XYZModem oder scp)
3. File System Check der MicroSD Karte
4. Crosskompilieren von Programmen für Gnublin
5. Test von Web-Anwendungen auf Gnublin mit einem Browser
6. Arbeiten mit dem Kernel

    - Kernelquelltext ändern
    - Quelltext hinzufügen, z.B. ein neues Modul
    - Suchen von Quelltextteilen

- Änderungen herausfinden (diff)
- Konfiguieren
- Kompilieren
- Kernel u. Module auf MicroSD Karte übertragen
- Versionsverwaltung mit git

7. Root Filesystem für Gnublin

   - Pakete hinzufügen/löschen
   - ELDK kompilieren
   - Andere Distributionen für Gnublin anpassen

Viele Anfänger sind geneigt, ausschliesslich mit grafischen "Klick-Werkzeugen" zu arbeiten. Ist das bei 1, 2 und 3 noch in Ordnung, gibt es bei 4, 5, 6 und 7 damit ernste Probleme, da alle verwendeten Programme auf der Kommandozeile angewendet werden. Es sieht frustrierend aus, wenn Anfänger lediglich mit einem grafischen Dateimanager ausgerüstet im Kernel Verzeichnisbaum verloren herumklicken. Um produktiv arbeiten zu können, lohnt es sich daher, zunächst zu lernen, wie man auf einem UNIX-ähnlichen Rechner effektiv auf der Kommandozeile arbeitet. Einige freie Literaturangaben dazu habe ich unter [1] gesammelt.

Die Projekte

1. Debian als Alternative zum ELDK

   http://forum.gnublin.org/viewtopic.php?f=4&t=45

2. I2C Real-Time Clock DS1307 zum Setzen der Systemzeit (Teil von 1.)
3. DCF77 Empfänger an GPIO Pin zum Setzen der Systemzeit
4. Sprachausgabe mit Soft-PWM + Tiefpassfilter an GPIO Pin

   In einem Kernelmodul wird eine PWM Frequenz von etwa 10 kHz erzeugt. Über eine Gerätedatei kann das PWM Puls- zu Pausenverhältnis geändert werden. Sprachsignale können aus einer Binärdatei mit 8-Bit Amplitudenwerten direkt durch "cat" auf die Gerätedatei ausgegeben. Die maximale Sprachfrequenz ist etwa 1 kHz.

5. Treiber für NEC Infrarot Protokoll
6. Mehrere LM75 Sensoren (I2C) an Gnublin
7. IMU 9 DoF (Sparkfun) an Gnublin mit interaktivem Web-Interface

   http://elk.informatik.fh-augsburg.de/hhwiki/EL12-Beschleunigung

8. Feuchte- und Temperatursensor SHT15 (I2C) an Gnublin

   Mit Hilfe des Sensors SHT15 (I2C) von Sensirion soll ein kombiniertes Feuchte- und Temperaturmessgerät entstehen.

9. 1-Wire

   Mit Hilfe eines DS2482 I2C auf 1-Wire Adapters soll Gnublin mit dem 1-Wire Bus sprechen.

10. GPS Modul GlobalTop GMS-D1

    Das GPS Modul aus dem Shop von Embedded Projects GmbH kostet nur etwa 15 Euro. Man kann es über einen UART oder über die Signale D+ und D- an USB anschliessen.

Eine genauere Beschreibung der Projekte wird es im Gnublin Wiki ab Mitte Juli geben.

Literatur

[1] H. Högl, Freie Literatur zu GNU/Linux und UNIX

http://elk.informatik.fh-augsburg.de/hhweb/doc/linuxbib.html