AREA-Unlimited

Bau einer ARM-Distribution

AREA 2.5

Nachfolgende Anleitung bildet einen Leitfaden zum Erstellen von bootfähigen Linux Images/SD-Karten für ARM-Boards basierend auf dem Allwinner A13 SoC. Als Development Board wurde ein OlinuXino A13 ohne WiFi verwendet (https://www.olimex.com/Products/OLinuXino/A13/A13-OLinuXino/).

Weiters wird kurz auf die Verwendung von Qemu und chroot eingegangen, um auf dem Host-System die Möglichkeit zu bieten das verwendet RootFS zu testen und mit den notwendigen Programmen auszustatten, falls es nicht möglich ist dies direkt auf dem Zielsystem zu erledigen.

Linux Grundkenntnisse werden vorausgesetzt und entsprechende Befehle nicht weiter erläutert.

Fabian EBENHOCH | Burkhard MARTIN v1.0

Übersicht

1. Versionsverwaltung

Listet die Änderungen einer Version zur nächsten auf.

1.1. Version 1.0

- Erstes Basissystem erstellt (auf Basis von Debian Wheezy)
- SPI voll implementiert
- GPIO voll implementiert
- Grundlegende Packete installiert
- X Server implementiert (mit XFCE als Window Manager)

2. Voraussetzungen

Um mit dem Build des Kernels/Systems fortfahren zu können sind nachfolgende Voraussetzungen zu erfüllen:

- Linux System oder Virtuelle Maschine mit Ubuntu (oder Debian wheezy)
- root/sudo - Rechte auf Host
- Schnelle Internetverbindung
- ca. 3GB Platz auf Host (1,5G Kernel/Uboot | 1,5 GB RootFS)
- Card-Reader (nur für SD-Karten Installation zwingend)

3. Installation peripherer Software

Zu Beginn wird die Toolchain aufgebaut, welche später verwendet wird um die einzelnen Dateien zu kompilieren, herunterzuladen etc.

Den ersten Schritt dazu stellt das hinzufügen neuer Paketquellen für apt-get dar. Dazu wird die sources.list Datei (zu finden unter /etc/apt/…) mit einem Texteditor geöffnet und folgende Zeilen hinzugefügt:

```
ACHTUNG: Muss als root editiert werden!
```

```
deb http://www.emdebian.org/debian/ wheezy main
deb http://www.emdebian.org/debian/ sid main
```

Anschließend kann mit dem aufsetzen der Toolchain begonnen werden.

```
# apt-get install gcc-4.6-arm-linux-gnueabi ncurses-dev uboot-mkimage
build-essential git
```

Der nächste Schritt besteht darin, eine symbolische Verlinkung auf den Compiler zu setzen, damit dieser später einfacher verwendet werden kann (Versionsnummer-unabhängig)

```
# ln -s /usr/bin/arm-linux-gnueabi-gcc-4.6 /usr/bin/arm-linux-gnueabi-
gcc.
```

4. Bau von U-boot

Den nächsten Arbeitsgang stellt das herunterladen und compilen von U-boot (für Infos siehe http://de.wikipedia.org/wiki/Das_U-Boot) dar.

Dazu wird im ersten Schritt ein neues Verzeichnis angelegt.

```
# cd /home/USERNAME
# mkdir olinuxino
# cd olinuxino
```

Nun laden wir uns das benötigte File für U-boot herunter und speichern es in unserem erstellten Verzeichnis

```
# git clone -b sunxi https://github.com/linux-sunxi/u-boot-sunxi.git
```

Danach sollte ein neues Verzeichnis existieren mit dem Namen der heruntergeladenen Datei vorliegen (in diesem Fall u-boot-sunxi). Nun wechseln wir in dieses Verzeichnis.

```
# cd u-boot-sunxi
```

Anschließend überprüfen wir die gemachten Schritte

```
# git checkout sunxi
```

Im nächsten Schritt wird die heruntergeladene Datei mittels CROSS_COMPILE auf die Zielarchitektur mit dem zu Beginn heruntergeladene Compiler kompiliert. Der Compiler wird über den Eingangs erstellen Symlink aufgerufen.

```
# make a13_olinuxino CROSS_COMPILE=arm-linux-gnueabi-
```

Um den letzten Arbeitsschritt zu überprüfen können wir folgenden Befehl verwenden:

```
# ls u-boot.bin spl/sunxi-spl.bin
```

Abschließend gehen wir wieder in unsren Hauptordner olinuxino:

```
# cd /home/USERNAME/olinuxino
```

5. Bau des Kernels

Nun kommen wir zum Hauptbestandteil unserer Arbeit, dem Kernel. Dafür laden wir als erstes das linux-sunxi (mehr Infos über linux-sunxi unter http://linux-sunxi.org/Main_Page) Verzeichnis herunter (und trinken einen Kaffee).

```
# git clone git://github.com/linux-sunxi/linux-sunxi.git
# cd linux-sunxi
# git checkout sunxi-3.4
```

Nach erfolgtem Download kreieren wir ein *.config* file.

```
# make ARCH=arm a13_defconfig
```

Um es wirkungsvoll bearbeiten zu können, öffnen wir es in einem externen Texteditor mit „root“-Rechten.

```
# sudo gedit .config
```

Im nun geöffneten File können Funktionen den Kernel hinzugefügt, oder auch entfernt werden. Beispielhaft hierfür werden die GPIOs eingebunden. Hierfür wird die unten dargestellte Zeile mit der darauffolgenden ersetzt.

```
1) CONFIG_SUN4I_GPIO_UGLY is not set

2) CONFIG_SUN4I_GPIO_UGLY=y
```

Damit haben wir unsre GPIO's aktiviert und das entsprechende Modul wird in den Kernel einkompiliert und somit beim booten mitgeladen. Soll allerdings nur das Modul erzeugt werden und nicht direkt eingebunden sein, sondern zur Laufzeit auf Wunsch des Users eingebunden werden, kann folgende Syntax verwendet werden:

```
CONFIG_SUN4I_GPIO_UGLY=m
```

Um SPI (Seriell Peripheral Interface) einzubinden ersetzen wir Zeile 1) mit Zeile 2).

```
1) CONFIG_SUN5I_SPI_UGLY is not set

2) CONFIG_SUN5I_SPI_UGLY=y
```

Nun kommen wir zur Kompilierung unsres Kernel's mit GCC. Hierfür einfach folgende Befehle in die Kommandozeile eingeben und ein bisschen Zeit mitbringen.

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- INSTALL_MOD_PATH=out
modules
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- INSTALL_MOD_PATH=out
modules_install
```

Ausgehend davon, dass keine Fehler auftauchten, ist die Kompilierung jetzt abgeschlossen.

6. Wahl des Betriebssystems

6.1. build your rootfs – kurzes HOW-TO auf Debian Wheezy Basis

Nachfolgend wird beispielhaft die Verwendung von debootstrap zur Erstellung eines RootFS auf Debian Wheezy Basis gezeigt.

Falls noch nicht installiert müssen folgende Packages installiert werden:

```
# apt-get install debootstrap qemu-kvm qemu-user-static
```

Im nächsten Schritt wird mittels debootstrap eine Hardfloat ARM Variante von Debian in den mit PFAD angegeben Ort gespeichert.

```
# debootstrap --verbose --foreign --arch=armhf --variant=minbase wheezy
/PFAD http://ftp.debian.org/debian/
# ln -sfn /PFAD/ /SYMLINK
```

Im nächsten Schritt werden die für die Emulation benötigen Daten in das heruntergeladene Image kopiert.

```
# cp /usr/bin/qemu-arm-static /SYMLINK /usr/bin
```

In den nachfolgenden beiden Schritten wird das RootFS entpackt und eingerichtet

```
# DEBIAN_FRONTEND=noninteractive DEBCONF_NONINTERACTIV_SEEN=true LC_ALL=C
LANGUAGE=C LANG=C chroot /SYMLINK /debootstrap/debootstrap --second-stage
# DEBIAN_FRONTEND=noninteractive DEBCONF_NONINTERACTIV_SEEN=true LC_ALL=C
LANGUAGE=C LANG=C chroot /SYMLINK dpkg --configure -a
```

Um eine einwandfreie Funktion in der chroot Umgebung zu garantieren werden mit folgenden Befehlen die entsprechenden Files vom Host auf das Zielsystem gemountet.

```
Diese Befehle sind bei jedem Neustart des Hosts und anschließender Ver-
wendung von chroot auszuführen!
```

```
# mount -t devpts devpts /SYMLINK/dev/pts
# mount -t proc proc /SYMLINK/proc
```

Um den Vorteile einer Internetverbindung in der chroot Umgebung zu nutzen muss noch der Inhalt des Verzeichnisses /etc/hosts auf das Zielsystem kopiert werden.

```
# cp /etc/hosts /SYMLINK/etc/
```

Um mit apt-get Programme installieren zu können müssen noch die Paketquellenauf dem Zielsystem richtig eingetragen werden. Nicht zu vergessen ist auch das setzen des Root-Passwortes, da ansonst das mühsam erstellte RootFS nicht zu gebrauchen ist.

```
# chroot /SYMLINK
# passwd ROOTPASSWORT
# export LC_ALL=C
```

Im nächsten Schritt werden die Paketquellen festgelegt. In der gewählten Variante „minbase“ ist KEIN Texteditor enthalten, so muss dies AUSSERHALB der chroot Umgebung erfolgen. Folgende Zeilen müssen im sources.list File (zu finden unter /etc/apt/) ergänzt werden:

```
ACHTUNG: Muss als root editiert werden!
```

```
deb http://ftp.debian.org/debian/ wheezy main contrib non-free
deb-src http://ftp.debian.org/debian/ wheezy main contrib non-free
```

Nun kann die Funktion von apt-get getestet werden:

```
# apt-get update
```

Nun können nach Herzenslust Programme, Services etc. heruntergeladen und installiert werden.

6.2. Herunterladen der Debian Distribution und script.bin

Alternativ zum eigens erstellten RootFS kann auch ein vorgefertigtes verwendet werden. Hier sind exemplarisch einige aufgelistet. Mittels wget werden die Files im aktuellen Ordner gespeichert.

```
# wget http://hands.com/~lkcl/mele_debian_armhf_minimal.cpio.gz
```

7. script.bin editieren

Zum booten des Systems wird neben uImage auch noch die Datei script.bin benötigt. Mittels folgendem Befehl kann dies heruntergeladen werden.

```
# wget
https://gist.github.com/raw/4090594/e661f202ea43496700f20238de66334a12922
4d0/script.bin
```

Um die .bin human-readable zu machen müssen folgende Befehle ausgeführt werden

7.1. Installation von bin2fex/konvertieren in script.fex und zurück

```
# cd /home/USERNAME/olinuxino
# git clone git://github.com/amery/sunxi-tools.git
```

```
# cd sunxi-tools
# make
# ./bin2fex /Gewünschter Pfad/script.bin / Gewünschter Pfad /script.fex
```

Die erzeugte .fex Datei kann nun mittels eines beliebigen Texteditor geöffnet werden und anschließend wieder in eine .bin Datei konvertiert werden

```
# sudo gedit script.fex
# ./fex2bin script.fex script.bin
```

8. Partitionierung der SD Karte

Nun führen wir folgenden Befehl aus, der uns alle angeschlossenen Laufwerke zeigt. Die Ausgabe merken wir uns, und vergleichen sie mit der Ausgabe des später folgenden Befehls.

```
# ls /dev/sd*
```

Anschließend stecken wir unsre SD Karte ein und führen den Befehl nocheinmal aus. Die neu dazu gekommenen Laufwerk/e ist/sind unsere SD-Karte.

Anschließend partitionieren wir unsere SD Karte neu. Das X muss durch den vorhandenen Laufwerksbuchstaben ersetzt werden.

```
# fdisk /dev/sdX
# p
```

Falls nun schon existierende Partitionen aufgelistet werden, löschen wir diese mit:

```
# d 1
# d 2
```

Immer mit ENTER bestätigen und anschließend noch einmal

```
# p
```

ausführen um zu kontrollieren ob die vorherigen Schritte erfolgreich waren.

Um die erste Partition zu erstellen, führen wir nun die folgenden Kommandozeilen aus.

```
# n
# p
# 1
# 2048
# 34815
```

Für die zweite Partition die folgenden Befehle ausführen:

```
# n
# p
# 2
# ENTER
# ENTER
```

Nun noch einmal

```
# p
```

um eine Tabelle der existierenden Partitionen zu sehen, in unserem Fall sollten es zwei sein, wie auf nachfolgendem Beispiel.

```
Disk /dev/sdd: 3980 MB, 3980394496 bytes
123 heads, 62 sectors/track, 1019 cylinders, total 7774208 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x7996dfb8

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1            2048       34815       16384   83  Linux
/dev/sdd2           34816     7774207     3869696   83  Linux
```

9. Schreiben der Dateien auf SD Karte

Abschließend schreiben wir alle bisher erzeugten Dateien auf die SD Karte. Das X muss wiederum durch den vorhandenen Laufwerksbuchstaben ersetzt werden.

```
# dd if=u-boot-sunxi/spl/sunxi-spl.bin of =/dev/sdX bs=1024 seek=8
# dd if=u-boot-sunxi/u-boot.bin of=/dev/sdX bs=1024 seek=32
```

9.1. Schreiben des Kernels und der script.bin Datei:

```
# mount /dev/sdX1 /mnt
# cp linux-sunci/arch/arm/boot/uImage /mnt
# cp script.bin /mnt
# sync
# umount /mnt
```

9.2. Schreiben der Distribution:

```
# mount /dev/sdX2 /mnt
```

Nun entweder das zuvor erzeugte RootFS auf die SD-Karte spielen oder das fertige, gepackte RootFS auf die SD-Karte entpacken.

Um das selbst erzeugte RootFS auf die SD-Karte zu spielen folgende Befehle verwenden:

```
# cd /SYMLINK
# cp -r * /mnt
```

Um das vorgefertigte RootFS zu entpacken folgende Befehle verwenden:

```
# cd /mnt
#  gunzip -c /home/USERNAME/olinuxino/mele_debian_armhf_minimal.cpio.gz |
cpio -i
```

Abschließend werden noch die Kernelmodules auf die SD-Karte geschrieben.

```
# cd /home/USERNAME/olinuxino
# cp -a linux-sunxi/out/lib/modules/KERNELVERSION/ /mnt/lib/modules
# sync
# umount /mnt
```

Die nun erzeugte SD-Karte ist, wenn alles richtig geklappt hat bootfähig und bietet je nach Variante die gewünschten Features.