

Ein ganzes GNU/Linux

Hans Buchmann FHNW/IME

5. Dezember 2016

Um was geht es ?

- ▶ ein GNU/Linux von Grund auf bauen
 - ▶ nicht mehr so schwer wie auch schon
- ▶ nicht völlig automatisiert
- ▶ Alternative zu **yocto** & Co.

Ziel

GNU/Linux auf dem BeagleBoneBlack

- ▶ command based
- ▶ Ethernet
- ▶ ssh
- ▶ sshfs
- ▶ moderne Toolchain inkl. *c++11* **C++**

Komponenten BeagleBoneBlack und *Host*

BeagleBoneBlack

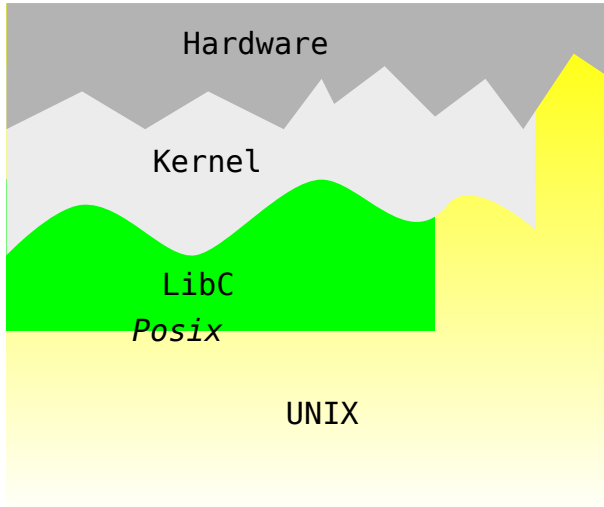
Kernel ein File

root ein Filesystem

Host

Toolchain binutils, gcc

Übersicht



Die Komponenten für BeagleBoneBlack

Hardware **BeagleBoneBlack**

Kernel zugeschnitten auf **BeagleBoneBlack**

- ▶ github.com/beagleboard/linux

root das Filesystem

LibC glibc

- ▶ www.gnu.org/software/libc/index.html

UNIX busybox

- ▶ www.busybox.net/

... Weitere UNIX basierte Komponenten

- ▶ das configure, make, make install
Triple

Toolchain

binutils linker & Co.

gcc compiler

- ▶ libgcc die Bibliothek für den Compiler

Remark(s):

- ▶ die Toolchain muss zweimal gebaut werden
 - ▶ für den **kernel** und libc
 - ▶ für UNIX/**POSIX**
- ▶ das target
 - ▶ cpu-vendor-os

Die Verzeichnisstruktur

```
├── tools
│   └── common.h .....used in (all) scripts
├── config
│   └── Makefile
├── build ..... home of the build files
├── target-root ..... top of target root
├── tc ..... the new toolchain
├── src ..... own programs
├── work
│   └── → ../config/Makefile
```


Build 1

Toolchain 1

- ▶ `binutils`
- ▶ `gcc-bare`
 - ▶ nur für den *kernel*
 - ▶ das bare minimum
 - ▶ nur `C`

Build 2

Kernel

- ▶ kernel mit ein paar *targets*
 - ▶ `bb.org_defconfig`
 - ▶ `zImage`
 - ▶ `headers_install`
 - ▶ Interface: *kernel-libc*

Build 3

libc

Wir brauchen eglibc

▶ eglibc

Build 4

Toolchain 2

- ▶ gcc
 - ▶ mit sysroot
 - ▶ C und C++
- ▶ Test
 - ▶ im Verzeichnis work

Build 5

busybox

- ▶ busybox
 - ▶ Installation auf SD-Card
 - ▶ fakeroot

Skripts und Argumente

initiales System

Skript	target	gebraucht für
binutils.sh		alles
gcc-bare.sh		kernel libc
kernel.sh	defconfig zImage headers_install	
glibc.sh		POSIX
gcc.sh		C/C++ POSIX
busybox.sh	busybox install	
target-root.sh		Transfer auf SD-Card

Remark: Alle Skripte sind **bash** Skripte

Target

erster Versuch

- ▶ transfer auf SD Karte
- ▶ Internet

Skripts und Argumente

ssh

zlib.sh

openssl.sh

die kryptographischen Algorithmen

openssh.sh

Remark: openssh.sh hängt von zlib.sh und openssl.sh ab

ssh

- ▶ openssh die volle Implementation
 - ▶ zlib
 - ▶ openssl
 - ▶ openssh

Workflow

Begriffe

- target-root** Verzeichnis auf dem *Host*
- ▶ enthält das **BeagleBoneBlack** Rootfilesystem
 - ▶ soll aktuell sein
- SD-Card** Speicherkarte mit dem **BeagleBoneBlack** Rootfilesystem
- ▶ entspricht `target-root`

target-root - SD-Card

tar rsync

	target-root		SD-Card
initiales GNU/Linux	→	tar	→
SD-Card	←	rsync	←
target-root	→	rsync	→

sshfs

funktioniert noch nicht

- ▶ Die Bibliothek `glib`
- ▶ Ersatz
 - ▶ `sftp`

Modules

- ▶ siehe 6-modules

Modules

Verzeichnisstruktur

Host

```
BUILD_HOME
├── modules
│   ├── Makefile ..... for own modules
│   └── *.c/h ..... the own sources
├── scripts
│   └── module.sh
```

BeagleBoneBlack

```
/
└── work
```

Herstellung Workflow

Host

- ▶ `sh scripts/module.sh module`
- ▶ `scp, sftp, ftp`

BeagleBoneBlack

- ▶ `insmod`
- ▶ `rmmmod`