

Teil I

Programmentwicklung

Ziele

Programmierung auf dem **BeagleBoneBlack**

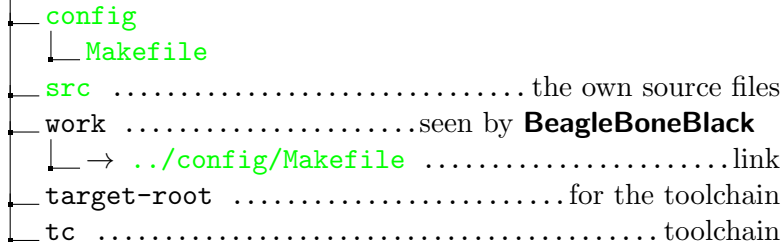
- ▶ (fast) wie auf dem *Host*
- ▶ *Toolchain* auf dem *Host*
- ▶ nur Programme (*runtime/executables*) auf dem **BeagleBoneBlack**
 - ▶ Sourcefiles bleiben auf dem *Host* (Ausnahme Skripts)
- ▶ Entwicklung für
 - C/C++ Posix *runtime*
 - Java Java SE Runtime Environment **BeagleBoneBlack**
 - ▶ nicht prioritär
 - Python Praktische platformunabhängige Sprache vom **BeagleBoneBlack** aus gesehen

Outline für C/C++

Host

dem **git** unterstellt

somewhere_on_your_host



BeagleBoneBlack

somewhere_on_your_**BeagleBoneBlack**

work mounted on *Host* per `sshfs`

Entwicklung

wo ist was

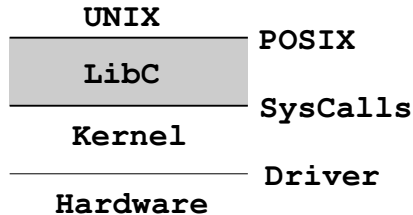
Host

- ▶ Toolchain/TargetRoot
`http://sourceforge.net/projects/fhnw-tinl/files/`
- ▶ Beispiele: `src/*`
- ▶ Herstellung: `make the-app`

BeagleBoneBlack

- ▶ Runtime GNU/Linux POSIX

POSIX → Kernel



POSIX `stdio.h` & Co

SysCalls → `target-root/usr/include/syscall.h`

Bibliotheken

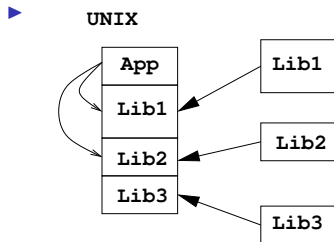
am Beispiel `hello-world-c.c`

- ▶ Der Objectfile `hello-world-c.o`
 - ▶ Der Code `objdump -d hello-world-c.o`,
 - ▶ Die Symbole `readelf -s hello-world-c.o`
- ▶ Das Image `hello-world-c`
 - ▶ Der Code `objdump -d hello-world-c`
 - ▶ Die Symbole `readelf -s hello-world-c.o`
- ▶ `puts`
 - ▶ ist in einer Bibliothek

Statische/Dynamische Bibliothek

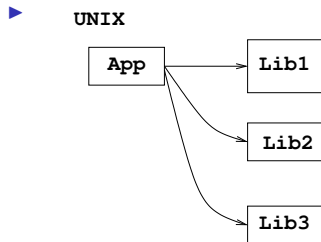
Kopie vs. Referenz

Static



► frühes Binden

Dynamic



► spätes Binden

→ copy
→ reference

Entwicklungsumgebung

- ▶ Entwicklungsumgebung aufsetzen
- ▶ Erste Programme
 - ▶ `hello-world-c.c`
- ▶ Minimale Programme
 - ▶ `direct-call.S`
 - ▶ `minimal-1.c` und `minimal-2.c` Makefile anpassen

Statische/Dynamische Bibliothek

- ▶ Die Programme
 - ▶ dynamisch linken
 - ▶ statisch linken
- und vergleichen
 - ▶ Grösse
 - ▶ objdump
 - ▶ readelf

Entwicklung

Plattformunabhängig

Host

- ▶ Toolchain sollte schon vorhanden sein
- ▶ Beispiel `HelloWorld.java`
- ▶ Herstellung `java -d. sourceFile`

BeagleBoneBlack

- ▶ Runtime `default-jre`

Aufgaben

- ▶ HelloWorld.java

Beachte `java -version, javac -source -target`

- ▶ Suche kleine Runtime
 - ▶ default-jre ist ziemlich gross
- ▶ Wie steht es mit
 - ▶ Oracle Java Platform, Micro Edition (Java ME)

Entwicklung <https://www.python.org/>
Plattformunabhängig

Host

- ▶ Entwicklungsumgebung (Editor)
- ▶ Viele nützliche Module

Batteries Included

- ▶ `src/hello-world.py`

BeagleBoneBlack

- ▶ Interpreter

Aufgaben

<https://github.com/adafruit/adafruit-beaglebone-io-python>

- ▶ Versuchen Sie GPIO mit Python

Teil II

Makefile

Programmentwicklung

von der *Source* zum *Image*

Gegeben: SourceFiles: viele Files

Gesucht: ImageFile: ein File

Programmentwicklung

Files sind die Grundelemente

- ▶ Klassische Programmentwicklung
- ▶ Verschiedene Arten von **Files**
- ▶ Programme/Tools erzeugen die Files
- ▶ Die Files hängen voneinander ab
- ▶ Für etwas komplexere Projekte gibt es viele Files ≈ 100

Ein grosses Projekt

GNU/Linux

- ▶ Anzahl Files
 - ▶ `tools/count-files.sh`
- ▶ SLOC: Source Line Of Code
 - ▶ `tools/sloc-count.sh`
 - ▶ Analyse mit z.B. *excel*

Der File Makefile

das Programm `make`

Makefile Muss selber geschrieben werden:
Beschreibt, wie Files gemacht werden.

Remark: Es gibt Programme z.B. automake die erzeugen Makefile's

make Programm:
interpretiert den `Makefile`

Dokumentation

<http://www.gnu.org/software/make/manual/make.html>

make: Aufruf

```
make name
```

make Das Programm

name Name des Files, der hergestellt werden soll ¹

Makefile muss nicht angegeben werden. `make` sucht den File mit dem Namen `Makefile` im *current directory*

► Alternative:

```
make -f path-to-makefile name
```

¹Allgemeiner: `name` ist der Name einer Regel

Makefile: Struktur

Variablen Siehe `→ config/Makefile`

Rules der wichtige Teil

Remark: Eine Regel beschreibt Abhängigkeiten

Makefile: *rule* Regel

```
target: → file1 file2 file3 ..  
        → tool
```

Remark: '→' steht für das unsichtbare *Tabulator* Zeichen

target File, der hergestellt wird

file1,file2.. *prerequisites* Files, die es braucht um das **target** herzustellen

tool Programm, das aus den *prerequisites* das **target** herstellt.

- Muss normalerweise nicht angegeben werden.
make kann aus den *Fileextensions* das **tool** bestimmen.

Ziel

Programme

1. lauffähig auf *Host*
2. lauffähig auf **BeagleBoneBlack**

dank **POSIX**

Verzeichnisstruktur

auf dem *Host*

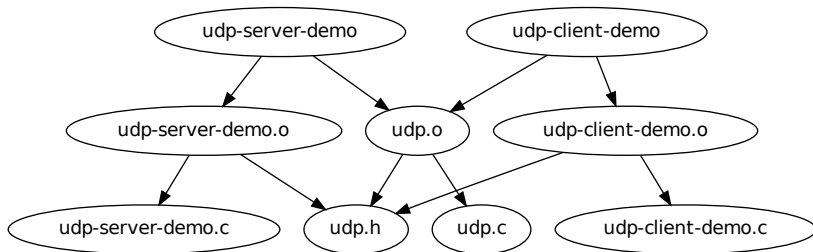
```

somewhere ..... on your Host
├── src ..... home of source files
├── config ..... home of configuration files
│   └── Makefile
└── work ..... workspace, connected with BBB
    └── Makefile → ../config/Makefile ..... link
```

Remark: Wie immer!

Abhängigkeiten

9 Files



Die Operationen

target	prerequisites	action
udp-server-demo:	udp-server-demo.o udp.o	link
udp-client-demo:	udp-client-demo.o udp.o	link
udp-server-demo.o:	udp-server-demo.c udp.h	compile
udp-client-demo.o:	udp-client-demo.c udp.h	compile
udp.o:	udp.c udp.h	compile

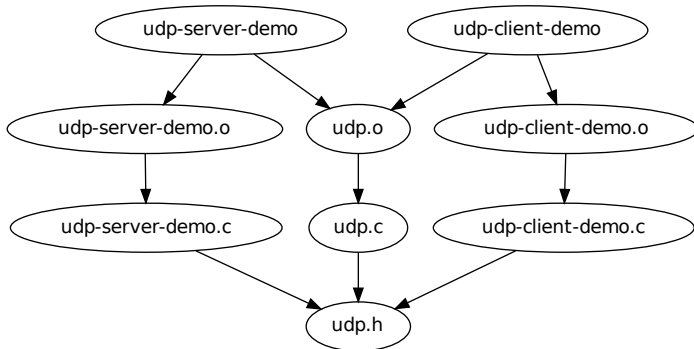
Die Include Files

Die *include files*:

- ▶ müssen im `Makefile` angegeben werden
- ▶ werden erst im vom Präprozessor inkludiert

Die Include Files

Andere Sichtweise



Aufgabe

- ▶ Anpassung an **BeagleBoneBlack**
- ▶ Für **BeagleBoneBlack** *und Host*
 - ▶ ist POSIX
- ▶ Nutzen Sie die *tools*
 - ▶ netcat
 - ▶ wireshark

Teil III

Crosscompiler

Hostrechner H

Targetrechner T

- ▶ Beispiel **BeagleBoneBlack**

Sourcefile `file.src`

- ▶ Beispiel `hello-world.c`

Executable `file(M)` ausführbar auf dem Rechner M ,
 $M = H|T$

- ▶ Beispiel `hello-world(T)`
für **BeagleBoneBlack**

$$\text{file.src} \rightarrow \boxed{\text{compiler}(M)} \rightarrow \text{file}(N)$$

file.src der Source File

compiler(*M*) der Compiler ein *executable* für den Rechner *M*

file(*N*) das *executable* für den Rechner *N*

Beispiel

Programm auf dem *Host*

`hello_world.src` \rightarrow `compiler(H)` \rightarrow `hello_world(H)`

```
gcc -O2 -std=c99 \  
../src/hello-world-c.c \  
-o hello-world-c
```

Beispiel

Crosscompilation

hello_world.src \rightarrow **compiler(H)** \rightarrow **hello_world(T)**

```
../tc/bin/arm-linux-gnueabi-gcc -O2 \  
--sysroot=../target-root -std=c99 \  
../src/hello-world-c.c \  
-o hello-world-c
```

`compiler.src` \rightarrow `compiler(H)` \rightarrow `compiler(H)`