Crossdevelopment

Hans Buchmann FHNW/ISE

5. November 2019

Entwicklung von Programmen auf dem **BeagleBoneWireless**

Nicht aus den Augen verlieren:

- alles ist ein File
  - 0 – *te* Näherung
  - File: *stream of bits*
  - wo sind die Files ?
- Filesysteme
  - `mount`
  - `sshfs`
- Cross development
  - *Host* $\leftrightarrow$ **BeagleBoneWireless**

Remark: Keine Toolchain auf dem **BeagleBoneWireless**

Wichtig

- wo ist was ?
  - Verzeichnisstruktur
- wo sind wir ?
  - *Host*
    oder
  - **BeagleBoneWireless**

Ein paar Befehle

- `cat` *name*
  - *concatenate files and print on the standard output*
- `hexdump -C` *name*
  - *display file contents in hexadecimal, decimal, octal, or ascii*
- `dd if=... of=... count=...`
  - *convert and copy a file*
- `cp`
  - *copy files and directories*
- `rsync`
  - *a fast, versatile, remote (and local) file-copying tool*
- `tar`
  - *archiving utility*

Devices sind auch Files

- SD-Karten `/dev/sd`**X**
- serielle Schnittstellen `/dev/tty`**X**
  - `/dev/ttyUSB0 /dev/ttyACM0`
- ...

Crossdevelopment

- zwei Rechner
  - Host der Entwicklungsrechner
  - Target **BBW** der Zielrechner
- Development
  - Wo sind die Files
- CrossDevelopment
  - Wo sind die Files

Outline

- Development
  - Programme auf dem *Host* für den *Host*
- CrossDevelopment
  - Programme auf dem *Host* für für den **BeagleBoneWireless**

Verzeichnisstruktur

```
6-crossdevelopment
├── src ..................................the source files
├── tc ..................... target toolchain normally link
├── target-work ............................. files for BBW
├── host-work ..............................files for Host
└── target-root .......copy from SD-card | link | mounted
```

Development (noch nicht Cross):Verzeichnis: `host-work`

die einzelnen Schritte

- Source file `src/hello-world.cc`
  - C++/POSIX

  unabhängig von Platform
- Object file (Maschinencode) `hello-world.o`
  - erzeugt mit: `g++ -c ../src/hello-world.cc -o hello-world.o`
  - Maschinencode:
    - `file hello-world.o`
    - `objdump -d hello-world.o`
- Executable file `hello-world`
  - `g++ hello-world.o -o hello-world`
  - Maschinencode:
    - `file hello-world-c`
    - `objdump -d hello-world-c`

In einem Schritt

für kleine Projekte

- `g++ ../src/hello-world.cc -o hello-world`

Was es braucht ?

Files

- Source file
  - wo ist der *include file* `iostream`
- Object File
  - `nm hello-world.o`
  - wo ist z.B. `_ZSt4cout`
- Executable
  - `nm hello-world`
  - `ldd hello-world`
  - wo sind die Bibliotheken

Wo sind die Files ?

irgendwo in einem Unterverzeichnis von /

- Include Files `g++ -v ../src/hello-world.cc -o hello-world`
  - `iostream` ?
- Bibliotheken
  - z.B. `libc.so`

Development

- Host==Target
- root Host==root Target

CrossDevelopment

- Host!=Target
- root Host != root Target

CrossDevelopment

Target **BeagleBoneWireless**

- toolchain
  - `tc/bin/arm-linux-gnueabihf-*`
  - `*`: `g++`,`nm`,`objdump` ...
- `target-root` Mehrere Möglichkeiten:
  - Kopie von SD-Karte
  - `sshfs debian@192.168.7.2:/ target-root`

CrossDevelopment: im Verzeichnis `target-work`

die einzelnen Schritte

- Source file `src/hello-world.cc`
  - C++/POSIX

  unabhängig von Platform
- Object file (Maschinencode) `hello-world.o`
  - erzeugt mit: `../tc/bin/arm-linux-gnueabihf-g++ --sysroot=../target-root -c ../src/hello-world.cc -o hello-world.o`
  - Maschinencode:
    - `file hello-world.o`
    - `../tc/bin/arm-linux-gnueabihf-objdump -d hello-world.o`
- Executable file `hello-world`
  - `../tc/bin/arm-linux-gnueabihf-g++ --sysroot=../target-root hello-world.o -o hello-world`
  - Maschinencode:
    - `file hello-world-c`
    - `../tc/bin/arm-linux-gnueabihf-objdump -d hello-world-c`

In einem Schritt

für kleine Projekte

- ```
  ../tc/bin/arm-linux-gnueabihf-g++ --sysroot=../target-root
  ../src/hello-world.cc -o hello-world
  ```
  ```

Unsere einfache Hardware

User LEDs

- Doku `tinL/doc/BBB_SRM.pdf` Abschnitt 6.6
- Zwei Arten
  - mit **BBW**`/sys/class/gpio/`
  - direkt mit `src/mem.h|cc` & Co

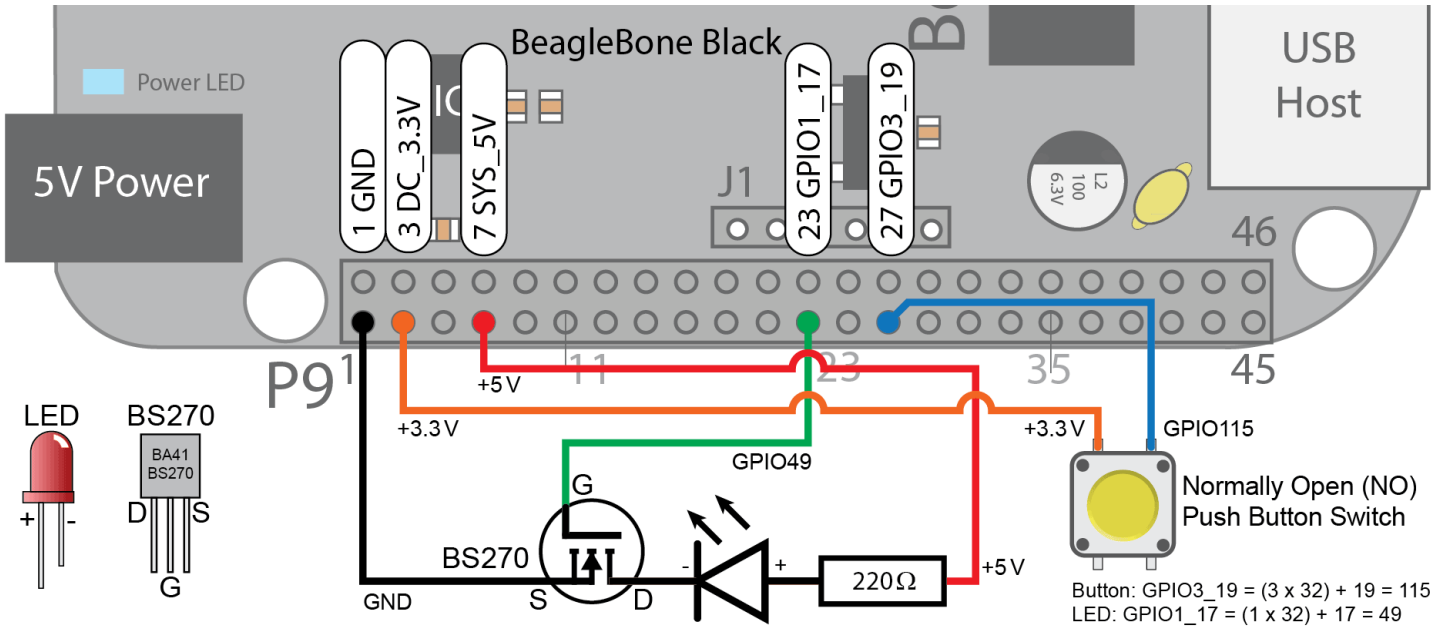
cod/sys/class/gpio/

- Kernel Sourcetree: `Documentation/gpio`
- Script
  - enable
  - blink
- C++
  - open/close
  - flush

Direkter Zugriff

- Hardware Dokumentation
  - `doc/beaglebone-black/spruh73l.pdf` Abschnitt 25
- Die Register:
  - `doc/beaglebone-black/spruh73l.pdf` 24.4.1
- der Code:
  - `led-direct-0.cc` als array
  - `led-direct-1.cc` als struct

Schema



©derekmolloy.ie/kernel-gpio-programming-buttons-and-leds

Ziel

- `hello-world` auf dem *Host* und auf dem **BBW**
- `primes` auf dem *Host* und auf dem **BBW**

The big Picture

- Source File: `hello-world.cc`
- falls es nicht klapt ?
  - wo ist der File ?

Scripts

- `led-enable.sh`
- `led-blink.sh`

C++

mit /sys/class/gpio

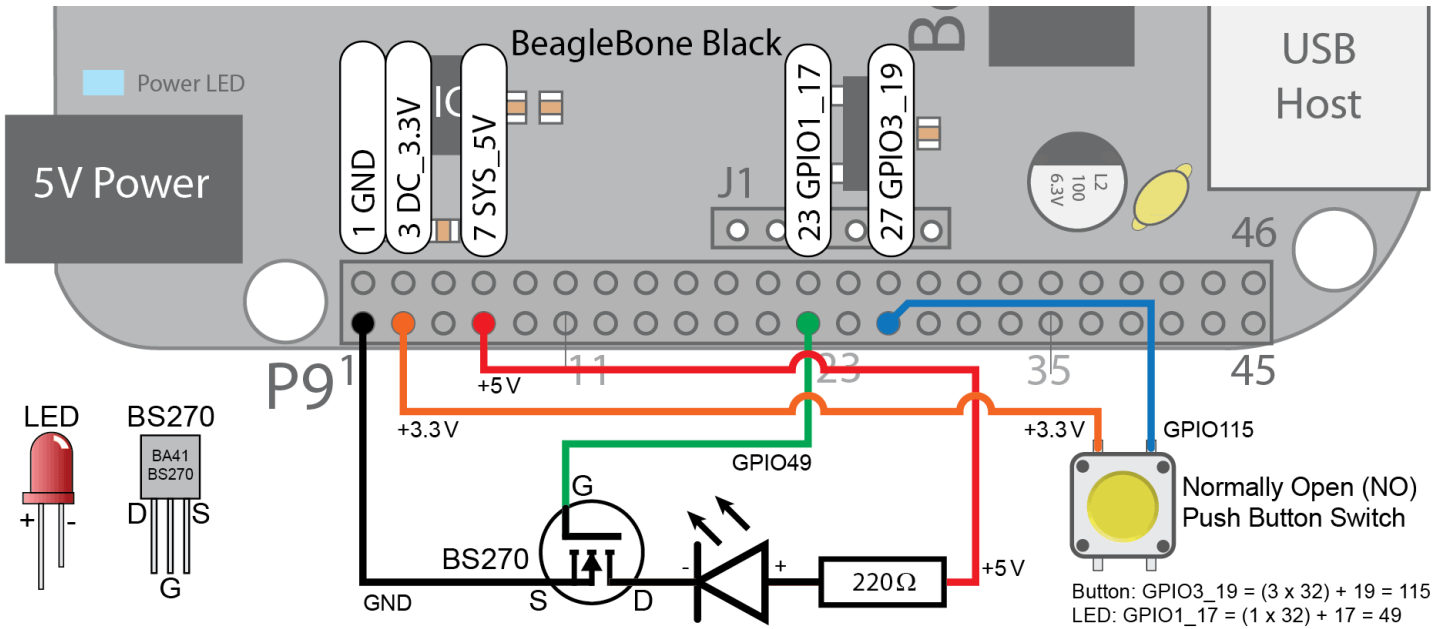
- led-enable.cc
- led-blink.cc

C++

direkt mit `mem.h|cc`

- `led-direct-0.cc`
- `led-direct-1.cc`

Input & Output



- Output LED (Script,C++)
- Input SWITCH (Script,C++)

Input & Output

Kombination

- SWITCH → LED
  - Script
  - C++

LED via Internet

Was Sie brauchen

- die LED Hardware
- einen HTTP Server
  - →lighttpd
  - den Code finden Sie auf →lighthttpd.tar.xz
- Common Gateway Interface (CGI)
  - `src/cgi0.cc` für den ersten Test
  - `src/cgi.h/cc` das *framework*
  - `src/cgi-test.cc` gfür den Test von `src/cgi.h/cc`
- Die Verbindung zur Hardware
  - `src/gpio.h/cc`

Die Herstellung

- `lighttpd`
  - Installation: `lighthttpd.tar.xz`
  - Konfiguration: `config/lighttpd.conf`
- CGI/GPIO (Makefile)
  - `src/cgi0.cc` für den ersten Test
  - `src/cgi.h/cc` das *framework*
  - `src/cgi-test.cc` gfür den Test von `src/cgi.h/cc`
  - `src/gpio.h/cc`