Embedded GNU/Linux Basics

Urs Fässler

17. LinuxDay
Dornbirn

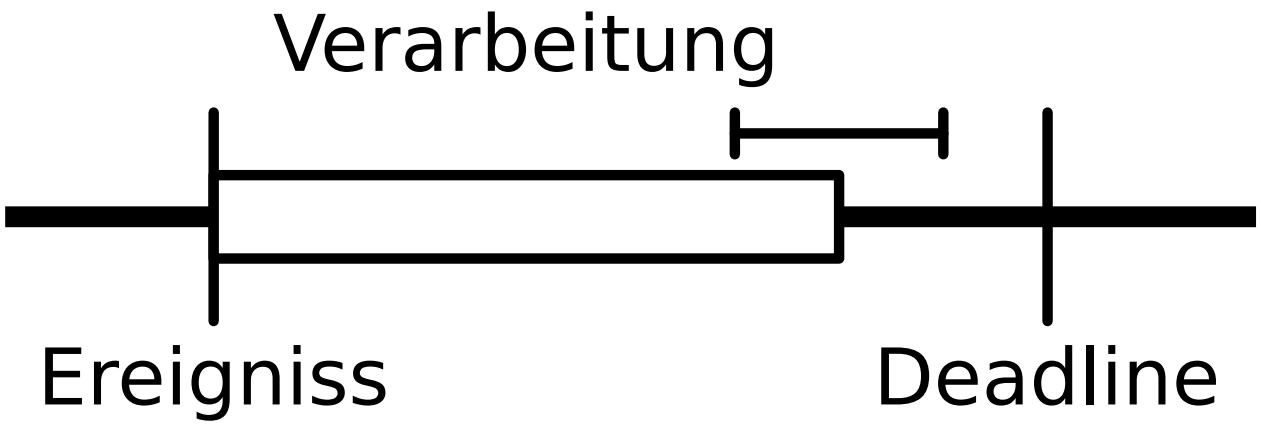
21.11.2015

Eigenschaften Embedded System

Eigenschaften Embedded System

- Kosten
- Grösse
- Ressourcen
- Energie
- Zuverlässigkeit
- Sicherheit
- Langlebigkeit
- Echtzeit

Echtzeit

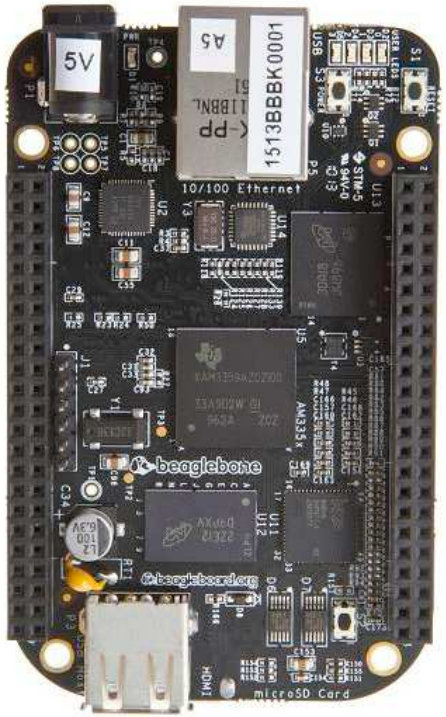


Definition Embedded System

Der Ausdruck embedded system bezeichnet einen Computer, der in einem technischen Kontext eingebettet ist.

Nach Wikipedia

Typisches GNU/Linux Embedded System



Aufgabe

Aufgabe

- Webserver für Most Useless Machine Ever!

Switch

property	value
switchOn	on
count	14

Servo

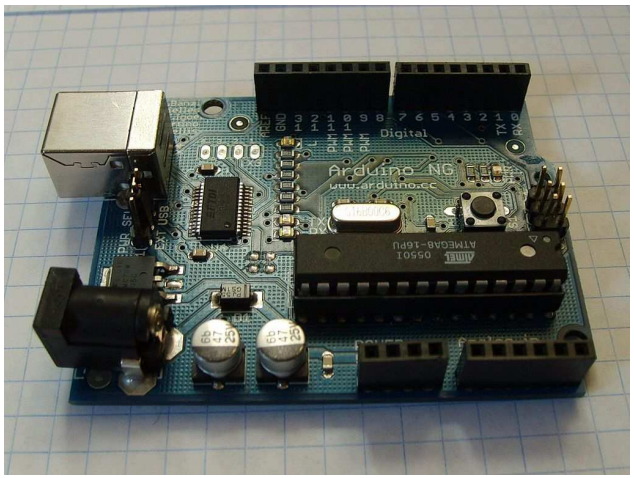
property	value	unit
open position	0.8	ms
close position	2.2	ms

Read Write

Vorgehen

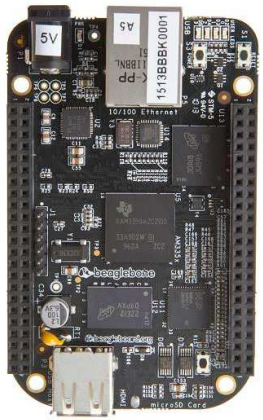
1. Hardware
2. OS
3. Image
4. SDK
5. Applikation
6. Deployment

Bare Metal uC



GNU/Linux SOC

Board



GNU/Linux Distribution

- off-the-shelf (Debian, OpenWRT, . . .)
- Yocto

GNU/Linux Distribution

- off-the-shelf (Debian, OpenWRT, . . .)
  - weite Verbreitung; bekannt
  - Updates werden von anderen bereitgestellt
  - erlaubt Lizenz Verteilung?
- Yocto

GNU/Linux Distribution

- off-the-shelf (Debian, OpenWRT, . . .)
  - weite Verbreitung; bekannt
  - Updates werden von anderen bereitgestellt
  - erlaubt Lizenz Verteilung?
- Yocto
  - git repository mit Konfiguration des gesamten System
  - Patchen einzelner Pakete
  - Optimierungen auf spezifische Hardware
  - volle Kontrolle

Yocto Image

mume-dev-image.bb

```
1  LICENSE = "MIT"
2
3  inherit core-image
4  inherit populate_sdk_qt5
5
6  IMAGE_INSTALL = " \
7    packagegroup-core-boot \
8    ...-core-ssh-openssh \
9    packagegroup-mume-common \
10   packagegroup-dev-mume \
11 "
12
13 IMAGE_FEATURES += " \
14   package-management \
15   debug-tweaks \
16 "
```

Yocto Image

mume-dev-image.bb

```
LICENSE = "MIT"

inherit core-image
inherit populate_sdk_qt5

IMAGE_INSTALL = " \
  packagegroup-core-boot \
  ...-core-ssh-openssh \
  packagegroup-mume-common \
  packagegroup-dev-mume \
"

IMAGE_FEATURES += " \
  package-management \
  debug-tweaks \
"
```

packagegroup-dev-mume.bb

```
SUMMARY = "developer tools
    for MUME"
LICENSE = "MIT"

inherit packagegroup

RDEPENDS_${PN} = "\
  bash \
  devmem2 \
  htop \
  nginx \
  openssh-sftp \
  perf \
  qtbase \
  time \
  ...
```

Yocto Image bauen

```
1 $ bitbake mume-dev-image
```

Yocto Image bauen

```
1 $ bitbake mume-dev-image
```

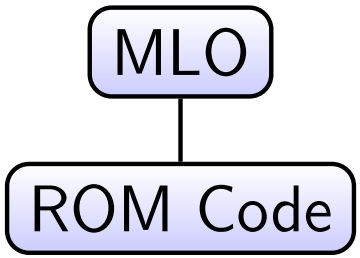
Table: tmp/deploy/images/beaglebone/

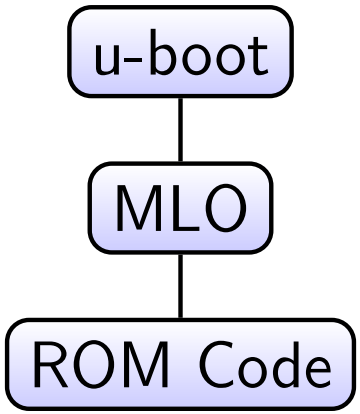
MLO	stage 1 loader
u-boot.img	stage 2 loader
uEnv.txt	u-boot Konfiguration
zImage	Kernel
zImage-bonegreen-mume.dtb	Device Tree
mume-dev-image-beaglebone.tar.bz2	rootfs

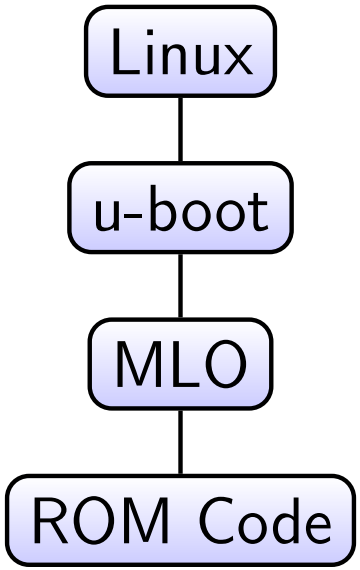
```
/
    bin
    boot
    etc
    home
    lib
    mnt
    opt
    root
    tmp
    usr
    var
    ...
```

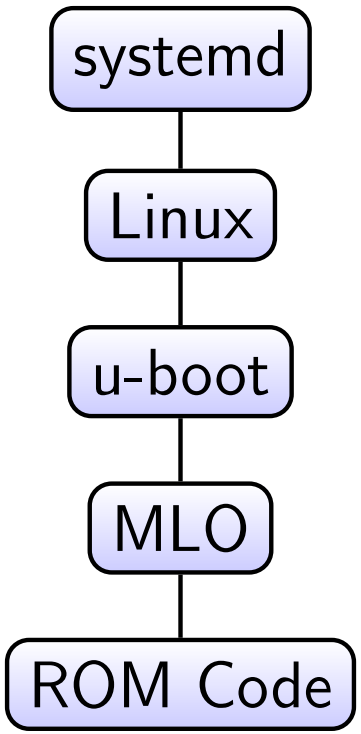
```
/ {
   compatible = "ti,am33xx";

   spi0: spi@48030000 {
      compatible = "ti,omap4-mcspi";
      status = "disabled";
      reg = <0x48030000 0x400>;
      interrupts = <65>;
      dmas = <&edma 16 &edma 17>;
      ...
   };
...
```

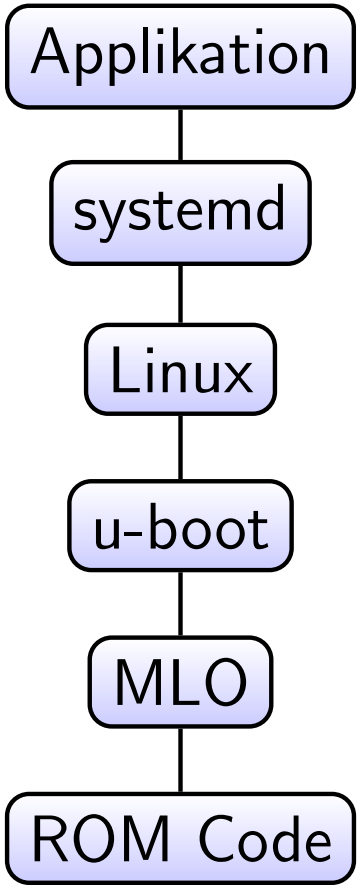
ROM Code


MLO
ROM Code


u-boot
MLO
ROM Code


Linux
u-boot
MLO
ROM Code


systemd
Linux
u-boot
MLO
ROM Code


Applikation
systemd
Linux
u-boot
MLO
ROM Code

SDK bauen

SDK bauen

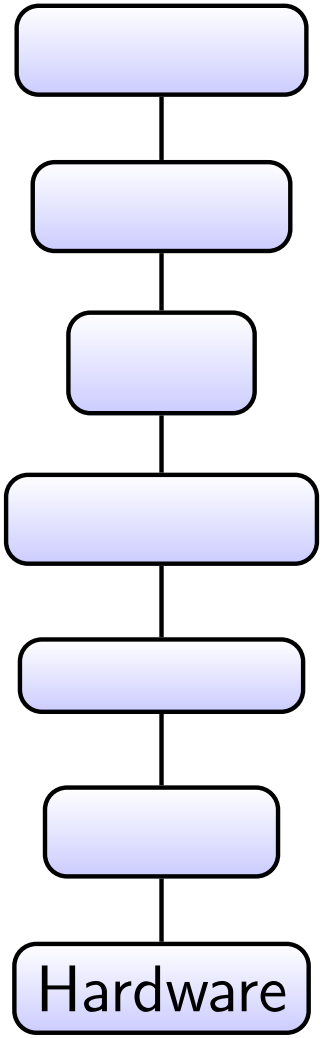
```
$ bitbake mume-dev-image -c populate_sdk
$ ls -sh tmp/deploy/sdk/
695M mume-glibc-x86_64-mume-dev-image-
   cortexa8hf-vfp-neon-toolchain-2.0.sh
```

SDK bauen

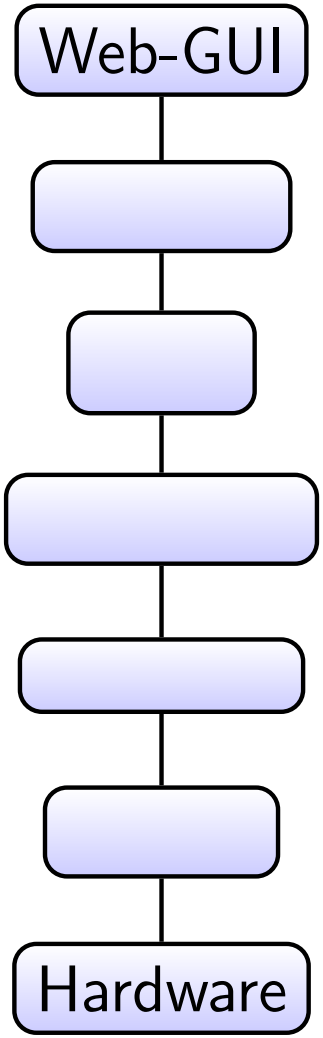
```
$ bitbake mume-dev-image -c populate_sdk
$ ls -sh tmp/deploy/sdk/
695M mume-glibc-x86_64-mume-dev-image-
   cortexa8hf-vfp-neon-toolchain-2.0.sh
```

- Entwicklerpakete (Header, ...)
- Cross-Compiler
- rootfs (Libraries, ...)
- Paketmanager

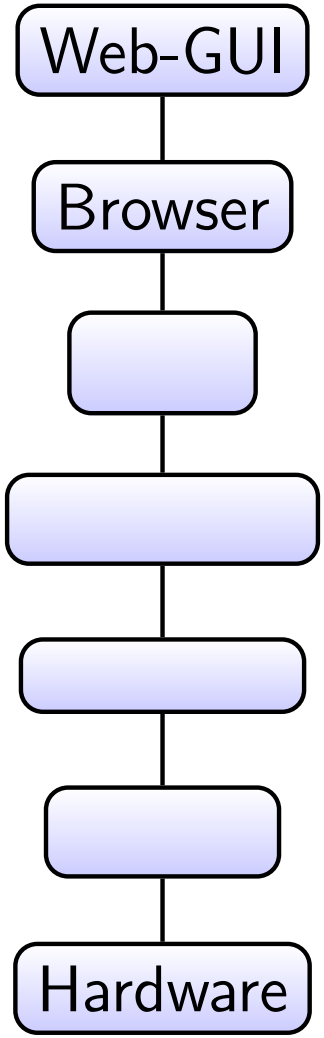
Übersicht



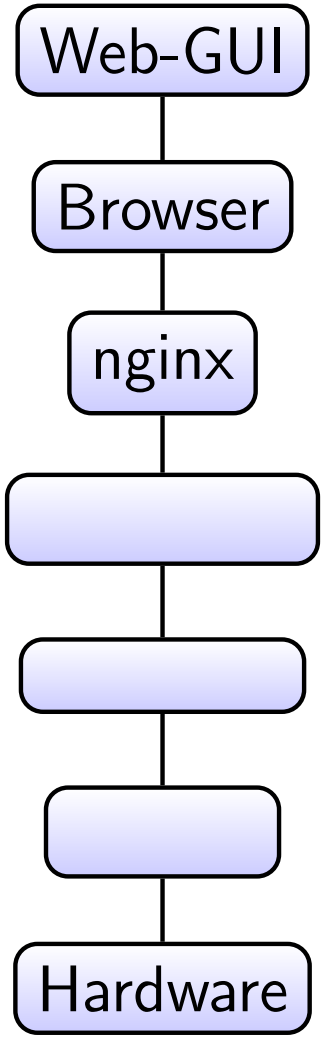
Übersicht



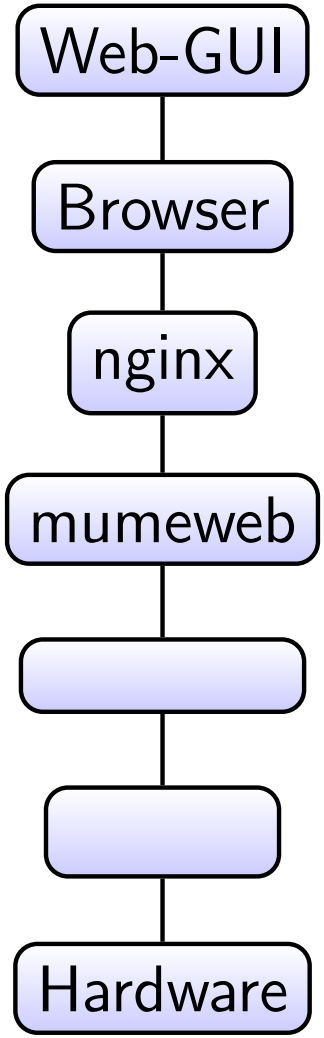
Übersicht



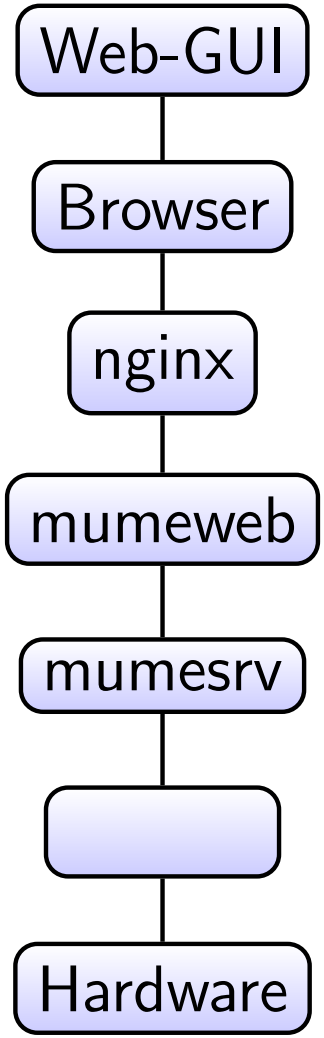
Übersicht



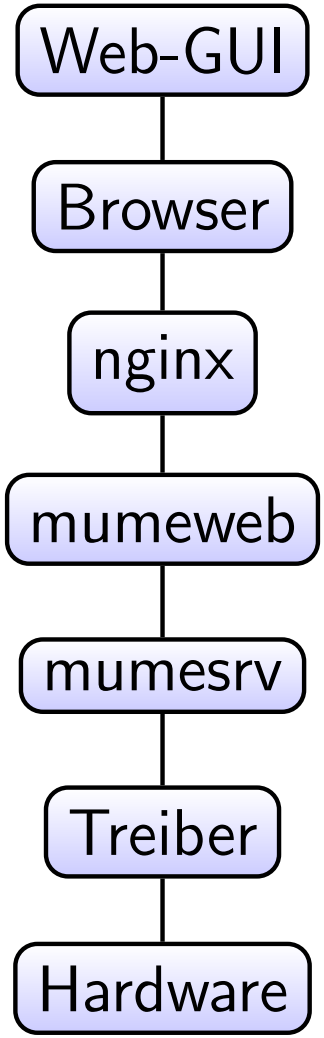
Übersicht



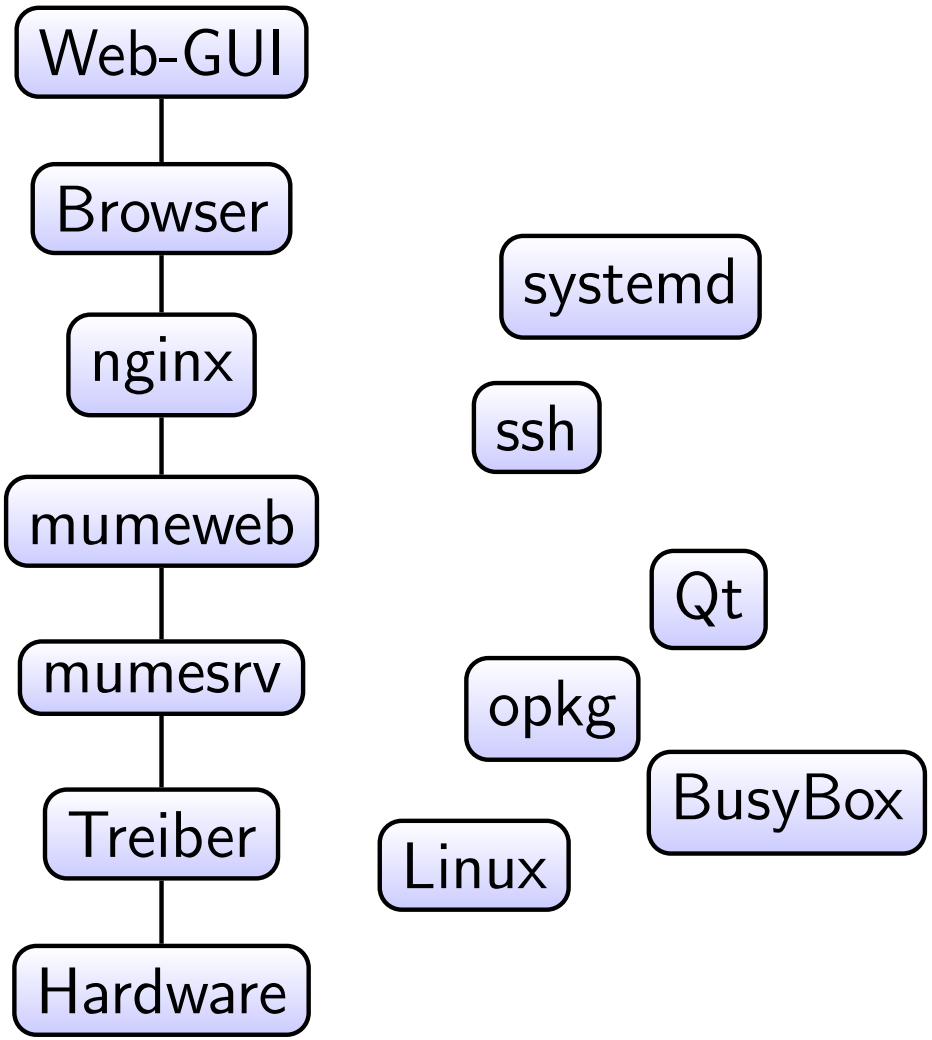
Übersicht



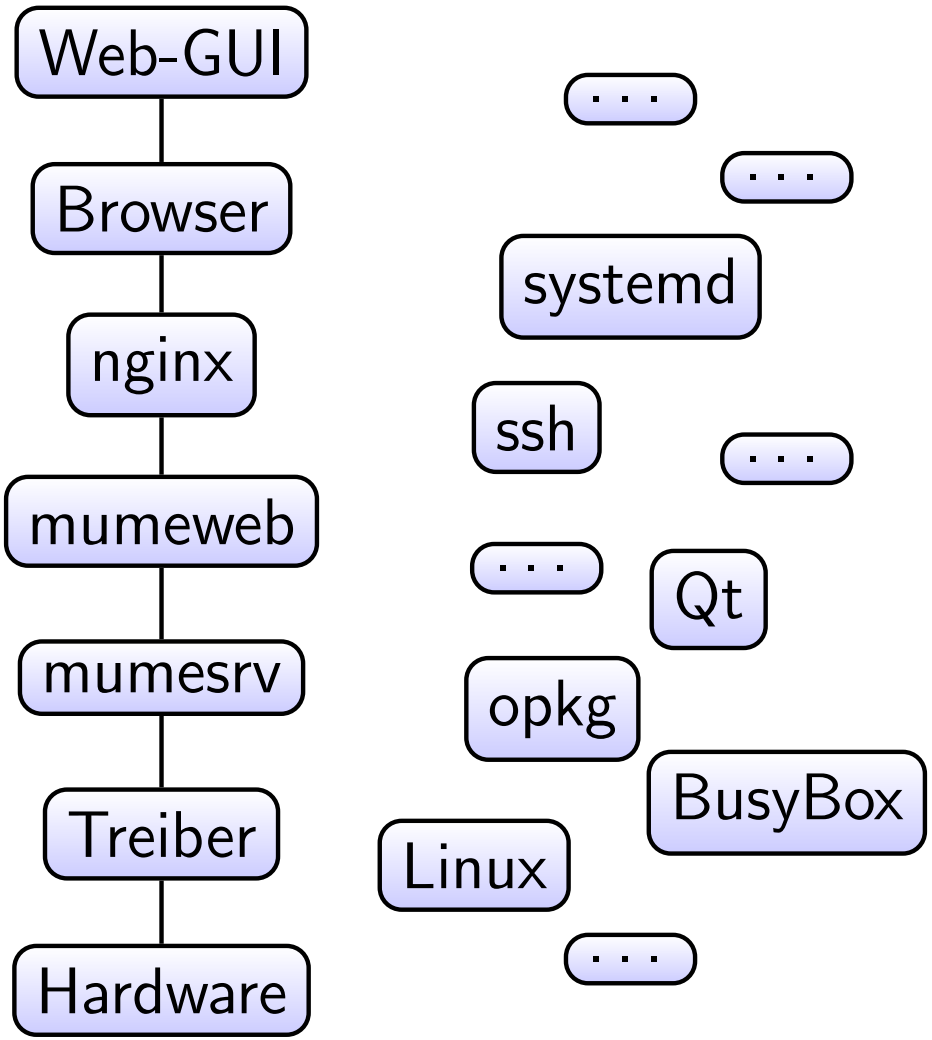
Übersicht

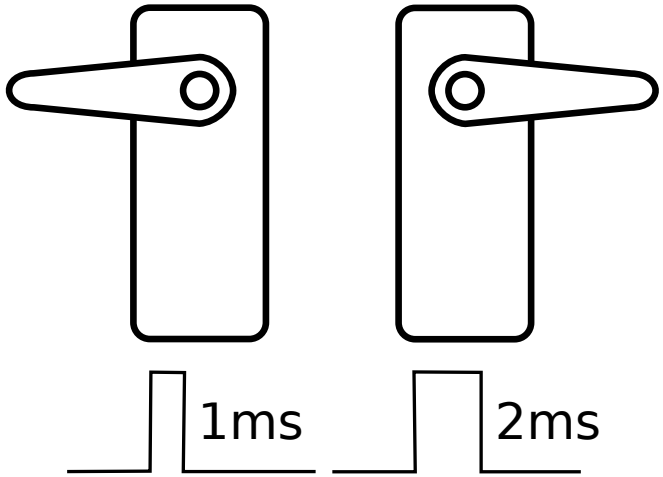


Übersicht


Web-GUI
Browser
nginx
mumeweb
mumesrv
Treiber
Hardware
systemd
ssh
Qt
opkg
BusyBox
Linux

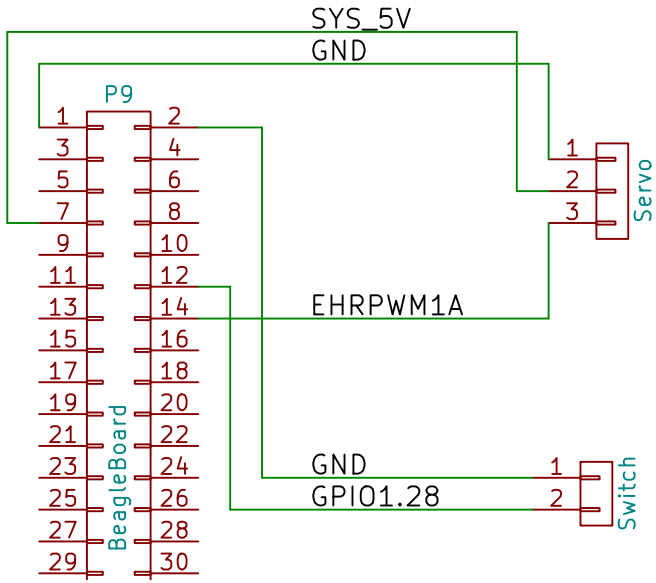
Übersicht


Web-GUI
Browser
nginx
mumeweb
mumesrv
Treiber
Hardware
...
...
systemd
ssh
...
...
Qt
opkg
BusyBox
Linux
...

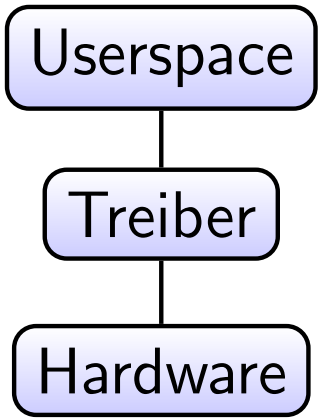

1ms
2ms

mume Device Tree

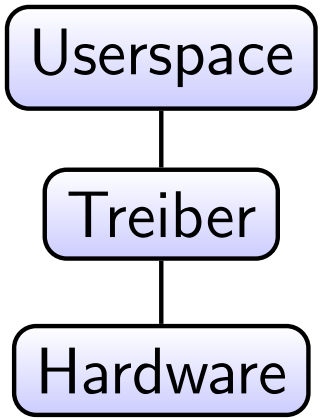
```
mume {
   compatible = "urs,mume";
   status = "okay";

   gpio = <&gpio1 28 GPIO_ACTIVE_LOW>;
   pwms = <&ehrpwm1 0 10000000 0>;

   pinctrl-names = "default";
   pinctrl-0 = <&mume_pins>;
};

mume_pins: mume_pins {
   pinctrl-single,pins = <
      0x78 (MUX_MODE7 | PIN_INPUT_PULLUP)
      0x48 (MUX_MODE6 | PIN_OUTPUT)
   >;
};
```

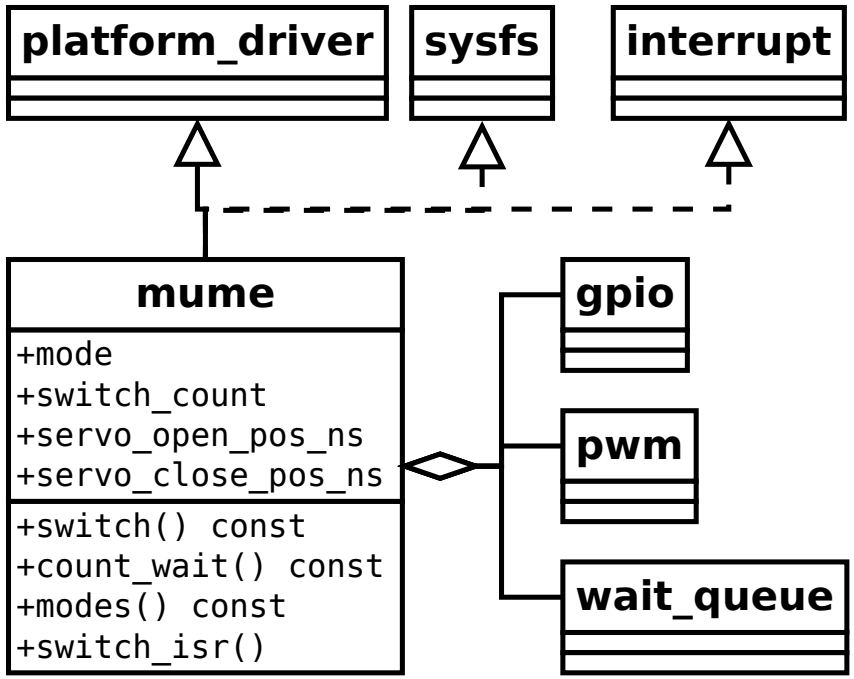


Hardware ansteuern

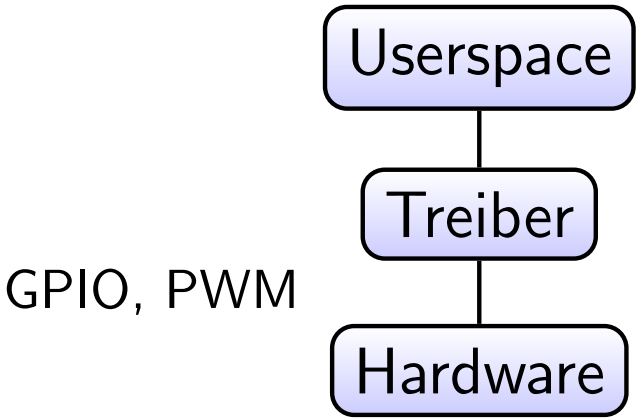


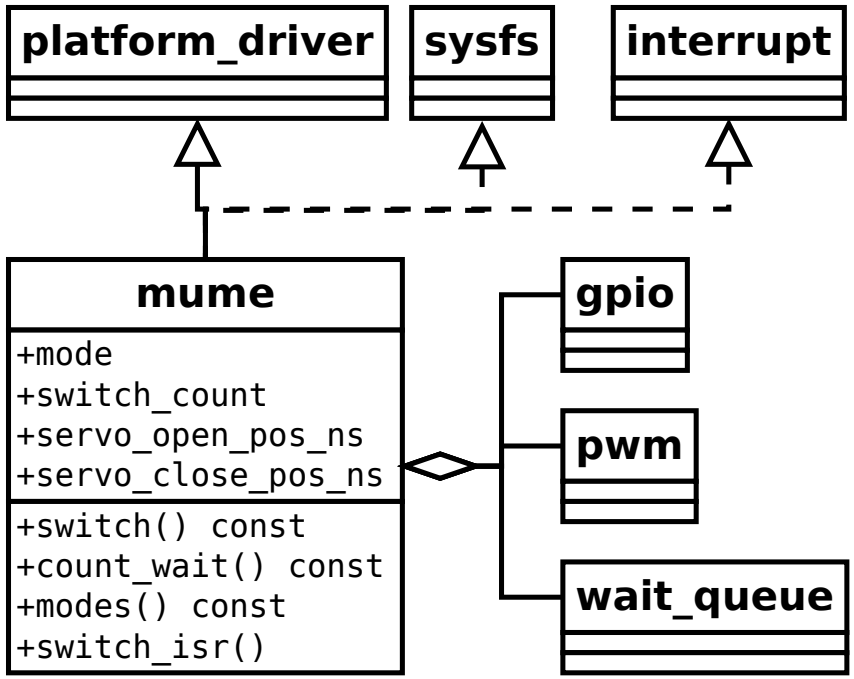
Hardware ansteuern



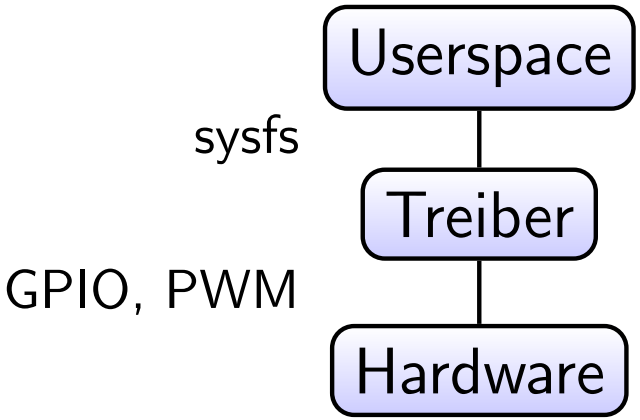


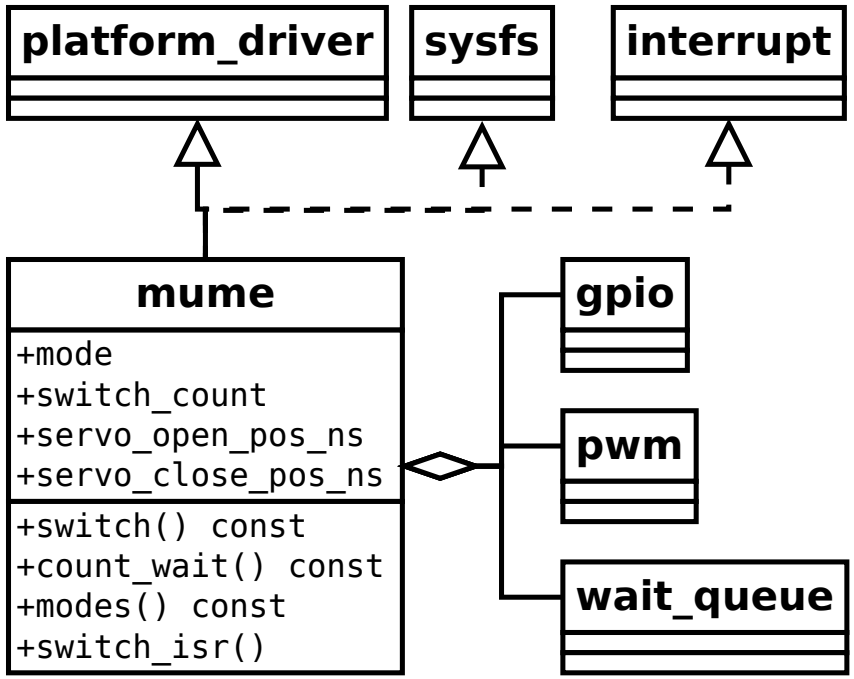
Hardware ansteuern



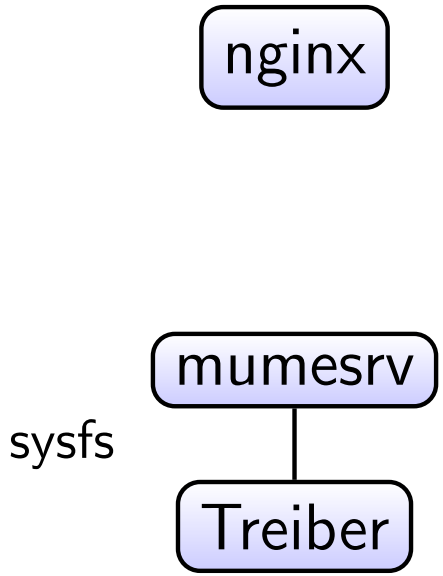


Hardware ansteuern

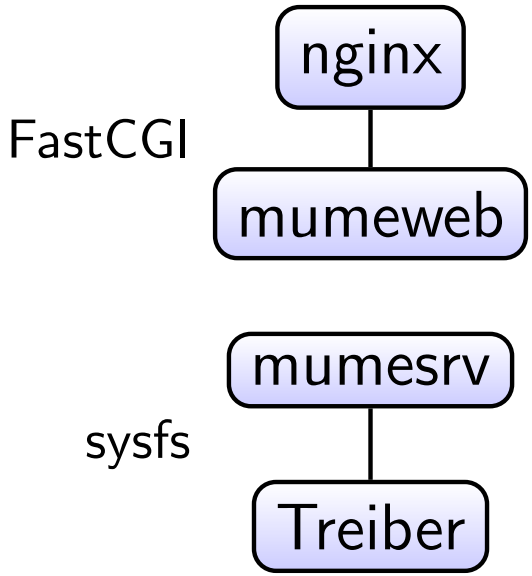




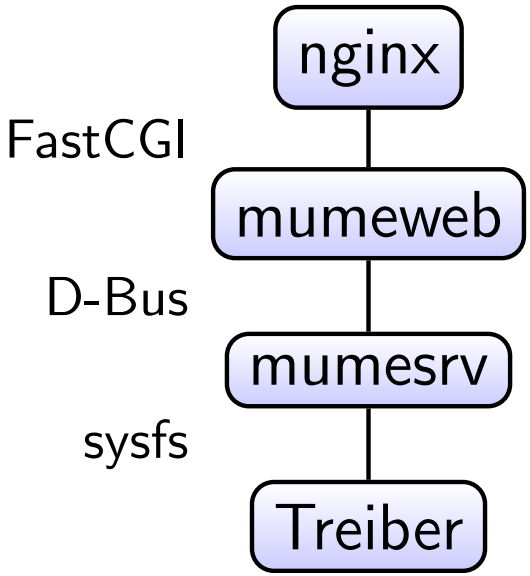

nginx
Treiber


nginx
mumesrv
sysfs
Treiber

vom Treiber zum Webserver



vom Treiber zum Webserver



Starten der Services

- systemd / unit file

```
1  [Unit]
2  Description=Mume D-Bus Service
3  After=dbus.target
4  Wants=dbus.target
5
6  [Service]
7  ExecStart=/usr/bin/mumesrv /sys/.../mume/
8
9  [Install]
10 WantedBy=multi-user.target
```

mumesrv Rezept

```
1  SECTION = "app"
2  DESCRIPTION = "MUME D-Bus interface to hardware"
3  LICENSE = "GPLv3+"
4  LIC_FILES_CHKSUM = "file://COPYING;md5=9
       eef91148a9b14ec7f9df333daebc746"
5
6  inherit qmake5
7
8  DEPENDS += "qtbase"
9  RDEPENDS_${PN} += "qtbase dbus"
10
11 SRCREV = "3cbe61c102d9574ccd039b112704f0a42a2112f8"
12
13 SRC_URI = " \
14         git://github.com/ursfassler/mumesrv.git;protocol=
               https;branch=master \
15         file://dbus.conf \
16 "
17
18 S = "${WORKDIR}/git"
19 QMAKE_PROFILES = "${S}/application/application.pro"
20 ...
```

Produktiv Image

mume-image.bb

```
1  LICENSE = "MIT"
2
3  inherit core-image
4
5  IMAGE_INSTALL = " \
6    packagegroup-core-boot \
7    packagegroup-core-ssh-openssh \
8    packagegroup-mume-common \
9    \
10   mumesrv-start \
11   mumeweb-start \
12   mumehtml \
13 "
14
15 IMAGE_FEATURES += " \
16   package-management \
17 "
```

Produktiv Image

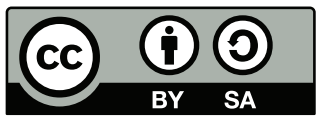
mume-image.bb

```
1  LICENSE = "MIT"
2
3  inherit core-image
4
5  IMAGE_INSTALL = " \
6    packagegroup-core-boot \
7    packagegroup-core-ssh-openssh \
8    packagegroup-mume-common \
9    \
10   mumesrv-start \
11   mumeweb-start \
12   mumehtml \
13 "
14
15 IMAGE_FEATURES += " \
16   package-management \
17 "
```

```
1  $ bitbake mume-image
```

Danke

- Slides auf `http://www.bitzgi.ch/presentations`
- Projekt auf `https://github.com/ursfassler`
  - /mumesrv Source von mumesrv
  - /mumeweb Source von mumeweb
  - /meta-mume Yocto Layer für Projekt, beinhaltet Linux Treiber als Patch
  - /embedded-gnu-linux-basics Sourcen des Vortrags



Urs Fässler urs@bitzgi.ch

Danke

- Slides auf `http://www.bitzgi.ch/presentations`
- Projekt auf `https://github.com/ursfassler`
  - /mumesrv Source von mumesrv
  - /mumeweb Source von mumeweb
  - /meta-mume Yocto Layer für Projekt, beinhaltet Linux Treiber als Patch
  - /embedded-gnu-linux-basics Sourcen des Vortrags

Urs Fässler urs@bitzgi.ch

Embedded GNU/Linux Basics

Urs Fässler

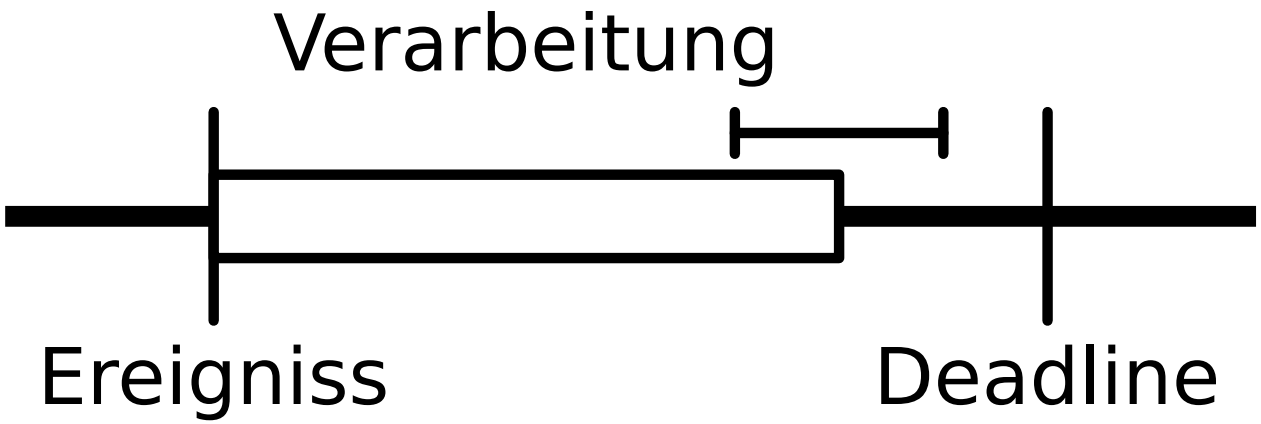
17. LinuxDay
Dornbirn

21.11.2015

Eigenschaften Embedded System[8]

- Kosten
- Grösse
- Ressourcen
- Energie
- Zuverlässigkeit
- Sicherheit
- Langlebigkeit
- Echtzeit

Echtzeit



- Bearbeitung ist nach einer bestimmten Zeit nach dem auftreten eines Ereignisses abgeschlossen.
- bei weicher Echtzeit ist dieses Verhalten wünschenswert (Video Wiedergabe)
- Echtzeit ist oft nicht nötig
- Linux ist nicht Echtzeit-Fähig
- Lösung ist separater uC, im SOC oder dediziertem Chip

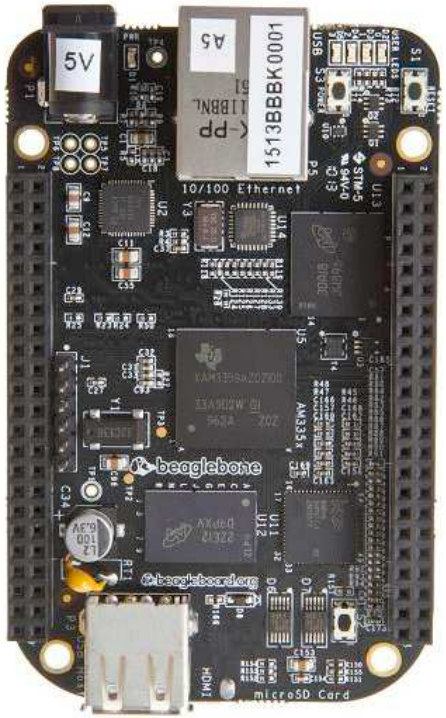
Definition Embedded System

Der Ausdruck embedded system bezeichnet einen Computer, der in einem technischen Kontext eingebettet ist. [15]

Nach Wikipedia

Typisches GNU/Linux Embedded System

[5]



[2]

Aufgabe

- Webserver für Most Useless Machine Ever!

[10]

Switch

property	value
switchOn	on
count	14

Servo

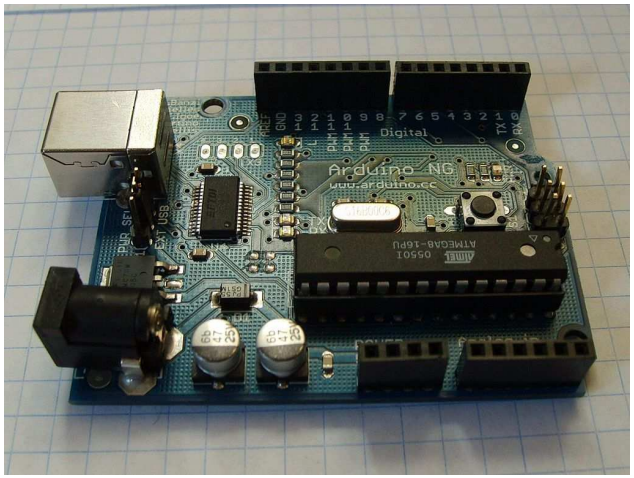
property	value	unit
open position	0.8	ms
close position	2.2	ms

Read Write

Vorgehen

1. Hardware
2. OS
3. Image
4. SDK
5. Applikation
6. Deployment

Bare Metal uC



Echtzeit
niedrige System-Komplexität
keine Infrastruktur

GNU/Linux SOC

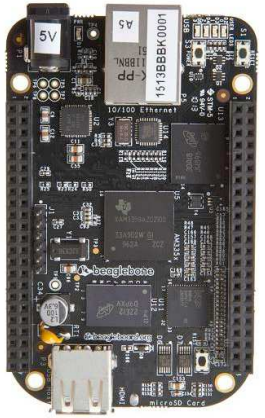
[5]

Memory und Prozess Management
Treiber & Protokolle
hohe System-Komplexität

Board


[7]

[4]



[2]

- Industrie-Boards
- Consumer Hardware (Router, Media-Center, . . .)
- Eval-/Bastelboards (Raspi, BeagleBone)

→ BeagleBone Green

- Netzwerk
- Yocto Supported
- viele Anshlüsse
- USB Powered

GNU/Linux Distribution

- off-the-shelf (Debian, OpenWRT, . . .)
  - weite Verbreitung; bekannt
  - Updates werden von anderen bereitgestellt
  - erlaubt Lizenz Verteilung?
- Yocto[3][1]
  - git repository mit Konfiguration des gesamten System
  - Patchen einzelner Pakete
  - Optimierungen auf spezifische Hardware
  - volle Kontrolle

[1] Tools und Rezepte um eigene GNU/Linux Distribution zu bauen

Yocto Image

mume-dev-image.bb

```
LICENSE = "MIT"

inherit core-image
inherit populate_sdk_qt5

IMAGE_INSTALL = " \
  packagegroup-core-boot \
  ...-core-ssh-openssh \
  packagegroup-mume-common \
  packagegroup-dev-mume \
"

IMAGE_FEATURES += " \
  package-management \
  debug-tweaks \
"
```

packagegroup-dev-mume.bb

```
SUMMARY = "developer tools
    for MUME"
LICENSE = "MIT"

inherit packagegroup

RDEPENDS_${PN} = "\
  bash \
  devmem2 \
  htop \
  nginx \
  openssh-sftp \
  perf \
  qtbase \
  time \
  ...
```

Yocto Image bauen

```
1 $ bitbake mume-dev-image
```

Table: tmp/deploy/images/beaglebone/

MLO	stage 1 loader
u-boot.img	stage 2 loader
uEnv.txt	u-boot Konfiguration
zImage	Kernel
zImage-bonegreen-mume.dtb	Device Tree
mume-dev-image-beaglebone.tar.bz2	rootfs

rootfs

```
/
   bin
   boot
   etc
   home
   lib
   mnt
   opt
   root
   tmp
   usr
   var
   ...
```

- userspace
- ev. Kernel & Device Tree
- nicht bootloader

Device Tree

```
/ {
   compatible = "ti,am33xx";

   spi0: spi@48030000 {
      compatible = "ti,omap4-mcspi"[2];
      status = "disabled"[3];
      reg = <0x48030000 0x400>[4];
      interrupts = <65>;
      dmas = <&edma[5] 16 &edma 17>;
      ...
   };
...
```

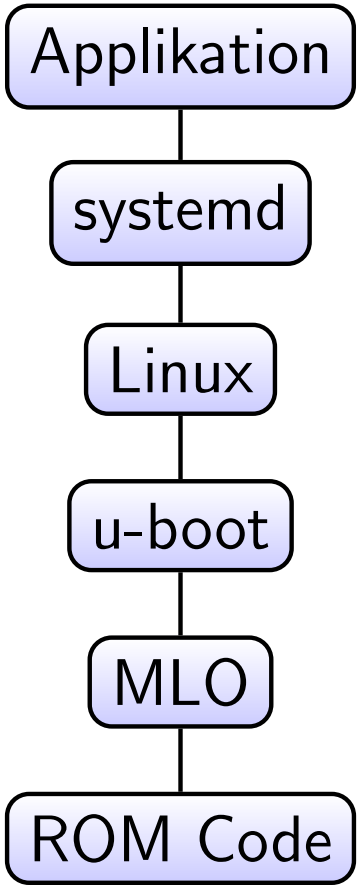
[2] Mapping zu Treiber
[3] aktivieren durch status="okay"
[4] Register-Position, siehe Datenblatt des SOC
[5] Referenz zum DMA Device-Tree Knoten

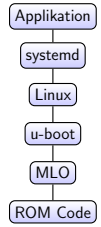
```
/ {
  compatible = "ti,am33xx";

  spi0: spi@48030000 {
    compatible = "ti,omap4-mcspi"[2];
    status = "disabled"[3];
    reg = <0x48030000 0x400>[4];
    interrupts = <65>;
    dmas = <&edma[5] 16 &edma 17>;
    ...
  };
...
```

[2] Mapping zu Treiber
[3] aktivieren durch status="okay"
[4] Register-Position, siehe Datenblatt des SOC
[5] Referenz zum DMA Device-Tree Knoten

- Linux kennt Hardware nicht
- Device Tree beschreibt Hardware
- Linux lädt Treiber anhand Device Tree

Boot Process[9]





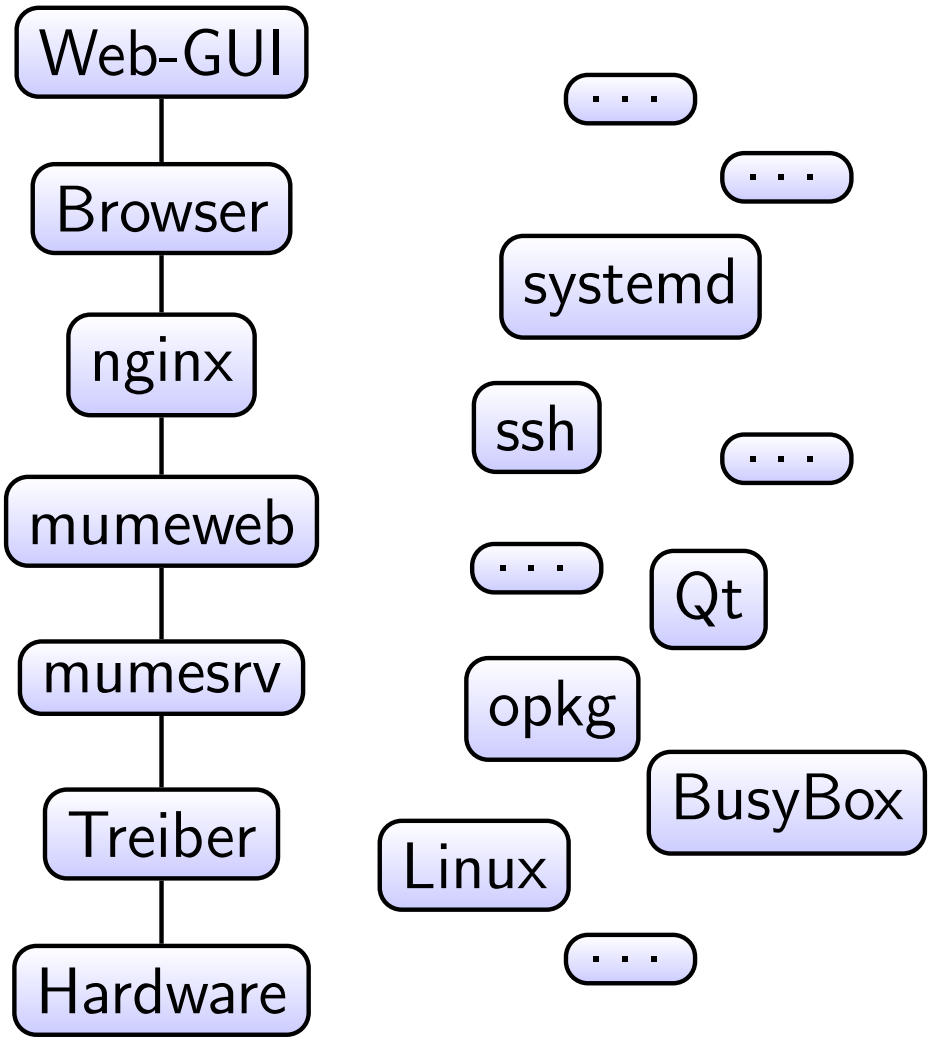
Typ	Name	Funktion
System startup	ROM Code	minimale Hardware Initialisierung in Boot-Devices nach Stage 1 Loader suchen Stage 1 Loader ins RAM laden und ausführen
Stage 1 Loader	MLO x-loader (u-boot)	Pin Muxing Clock und Memory initialisieren Stage 2 Loader ins RAM laden und ausführen
Stage 2 Loader	u-boot.img	Plattform Initialisierung (USB, Netzwerk, . . .) Boot-Menü / Kommandozeile anzeigen Kernel und Device-Tree ins RAM laden und ausführen
Kernel	zImage Linux	Treiber für Hardware laden rootfs mounten Init-Process starten
Init	Systemd	Abhängigkeiten zwischen Services auflösen Services starten Services überwachen

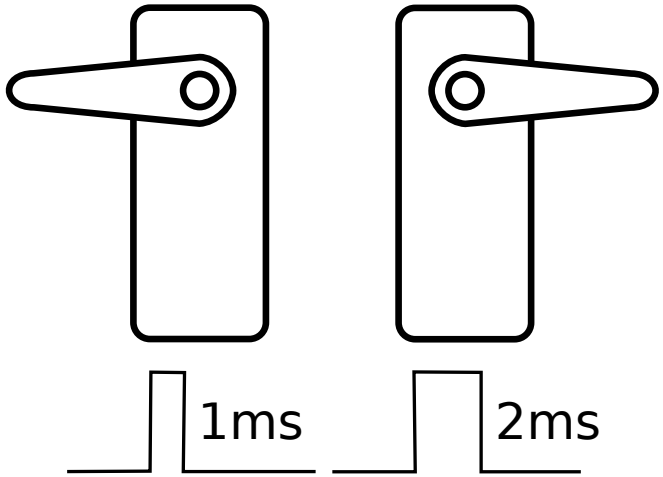
SDK bauen

```
$ bitbake mume-dev-image -c populate_sdk
$ ls -sh tmp/deploy/sdk/
695M mume-glibc-x86_64-mume-dev-image-
   cortexa8hf-vfp-neon-toolchain-2.0.sh
```

- Entwicklerpakete (Header, ...)
- Cross-Compiler
- rootfs (Libraries, ...)
- Paketmanager

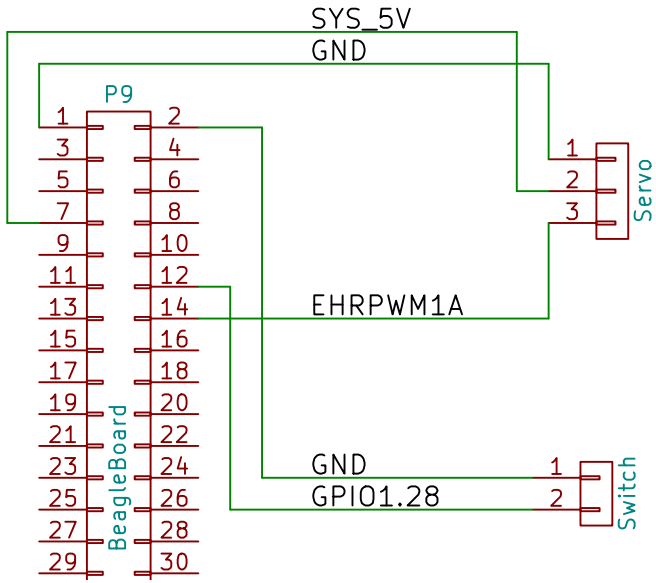
Übersicht


Web-GUI
Browser
nginx
mumeweb
mumesrv
Treiber
Hardware
...
...
systemd
ssh
...
...
Qt
opkg
BusyBox
Linux
...

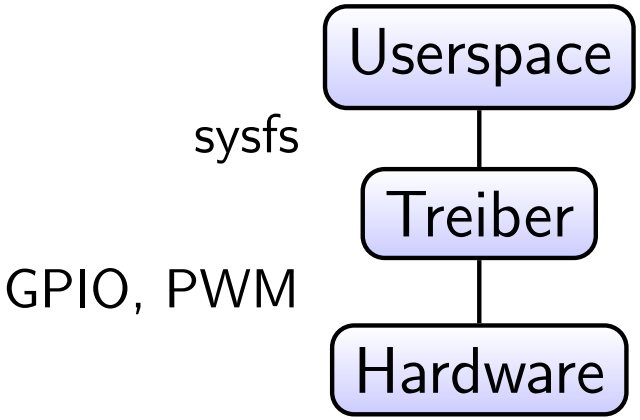

1ms
2ms

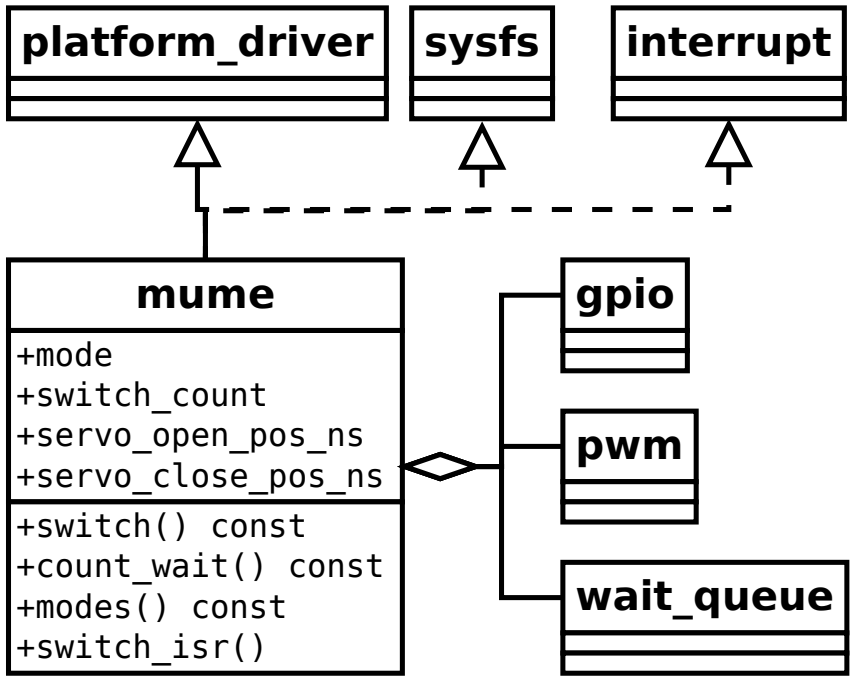
mume Device Tree

```
mume {
   compatible = "urs,mume";
   status = "okay";

   gpio = <&gpio1 28 GPIO_ACTIVE_LOW>;
   pwms = <&ehrpwm1 0 10000000 0>;

   pinctrl-names = "default";
   pinctrl-0 = <&mume_pins>;
};

mume_pins: mume_pins {
   pinctrl-single,pins = <
      0x78 (MUX_MODE7 | PIN_INPUT_PULLUP)
      0x48 (MUX_MODE6 | PIN_OUTPUT)
   >;
};
```



Hardware ansteuern



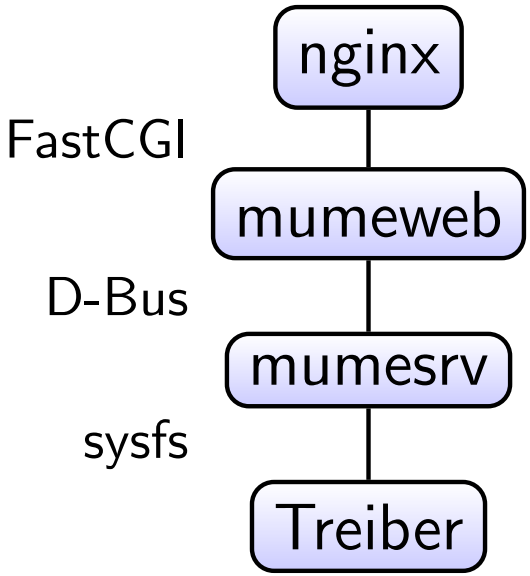


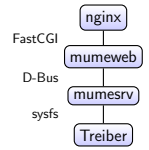
- wie können wir die Hardware ansteuern?

→ Linux Treiber, Frameworks, Infrastructure:

- Framework für Treiber
- Infrastrukur um auf Subsysteme zuzugreifen

vom Treiber zum Webserver





- mumeweb
  - FastCGI / XML Interface
  - (Session Handling)
- mumesrv
  - Persistenz
  - D-Bus Interface

Starten der Services

- systemd / unit file

```
1  [Unit]
2  Description=Mume D-Bus Service
3  After=dbus.target
4  Wants=dbus.target
5
6  [Service]
7  ExecStart=/usr/bin/mumesrv /sys/.../mume/
8
9  [Install]
10 WantedBy=multi-user.target
```

mumesrv Rezept

```
1  SECTION = "app"
2  DESCRIPTION = "MUME D-Bus interface to hardware"
3  LICENSE = "GPLv3+"
4  LIC_FILES_CHKSUM = "file://COPYING;md5=9
       eef91148a9b14ec7f9df333daebc746"
5
6  inherit qmake5
7
8  DEPENDS += "qtbase"
9  RDEPENDS_${PN} += "qtbase dbus"
10
11 SRCREV = "3cbe61c102d9574ccd039b112704f0a42a2112f8"
12
13 SRC_URI = " \
14         git://github.com/ursfassler/mumesrv.git;protocol=
               https;branch=master \
15         file://dbus.conf \
16 "
17
18 S = "${WORKDIR}/git"
19 QMAKE_PROFILES = "${S}/application/application.pro"
20 ...
```

Produktiv Image

mume-image.bb

```
1  LICENSE = "MIT"
2
3  inherit core-image
4
5  IMAGE_INSTALL = " \
6    packagegroup-core-boot \
7    packagegroup-core-ssh-openssh \
8    packagegroup-mume-common \
9    \
10   mumesrv-start \
11   mumeweb-start \
12   mumehtml \
13 "
14
15 IMAGE_FEATURES += " \
16   package-management \
17 "
```

```
1  $ bitbake mume-image
```

mume-image.bb

```
LICENSE = "MIT"

inherit core-image

IMAGE_INSTALL = " \
  packagegroup-core-boot \
  packagegroup-core-ssh-openssh \
  packagegroup-mume-common \
  \
  mumesrv-start \
  mumeweb-start \
  mumehtml \
"

IMAGE_FEATURES += " \
  package-management \
"
```

```
$ bitbake mume-image
```

- Wie können wir die Firmware verteilen?

→ Yocto

  - Rezepte für Services
  - anpassen bestehender Rezepte (nginx, qt, . . .)
  - Rezept für Image

- bitbake image

Danke

[12]

[13]

- Slides auf `http://www.bitzgi.ch/presentations`
- Projekt auf `https://github.com/ursfassler`
  - /mumesrv Source von mumesrv
  - /mumeweb Source von mumeweb
  - /meta-mume Yocto Layer für Projekt, beinhaltet Linux Treiber als Patch
  - /embedded-gnu-linux-basics Sourcen des Vortrags

Urs Fässler urs@bitzgi.ch

Embedded Systeme

Digitales Multimeter

[6]

- Sensor produziert 20 B/sec
- Reduzierung auf 2 B/sec
- Verarbeitung auf internem uC

ATLAS/LHC/CERN

[1]

- Sensor produziert 1 PiB/sec
- Reduzierung auf 100 MiB/sec [14]
- Verarbeitung auf eigenen 20'000 Server und Grid [11]

Firmware

"Unter Firmware versteht man Software, die in elektronische Geräte eingebettet ist." [16]

- Gesamtes Software-Image von Embedded System
- Software für Subsysteme
  - Power Management
  - WLAN Karte
  - BIOS
  - Touch-Screen

PID 1

Nachdem Linux alles initialisiert hat wird Kontrolle an Userspace übergeben. Üblicherweise ist dies systemd.

- systemd
  - einfach da bekannt
  - wenn mehrere Dienste nötig
  - gewisse Grösse
- Script
  - wenn nur wenige Dienste
- Applikation
  - Applikation muss alles machen
  - nur für monolithische Applikationen

Aufgaben um Embedded system zu erstellen

- Cross Compiler bauen
- sysfs bauen (Kernel und benötigte Tools für System)
- Cross Compiler und sysfs für Developer bereitstellen (SDK)

Aufgaben um Embedded system zu deployen

- Applikation cross-compilen
- sysfs cross-compilen
- Kernel cross-compilen
- Applikationen paketieren
- Image zusammenstellen

- Verteilen von Ressourcen
- Initialisieren und Abstrahieren der Hardware

Sonstiges

busybox Reimplementation von GNU/Linux Tools in einem Binary, optimiert auf groesse; oft eingeschraenker Funktionsumfang

important to know

opkg -o /opt/mume/1.8.1/sysroots/cortexa8hf-vfp-neon-poky-linux-gnueabi/ install /projekte/yocto/mume/build/tmp/deploy/ipk/cortexa8hf-vfp-neon/libtinyxml*

Literature I

Literatur & Bild-Nachweise

[1] Argonne National Laboratory.
cc-by-sa-2.0.
URL: https://commons.wikimedia.org/wiki/File:ATLAS_Tile_Calorimeter.png.

[2] BeagleBoard.org Foundation.
cc-by-sa-3.0.
URL: https://commons.wikimedia.org/wiki/File:Beaglebone_Black.jpg.

[3] davest.
Why embedded linux needs a project like the yocto project, 11 2010.
URL: https://www.yoctoproject.org/blogs/davest/2010/why-embedded-linux-needs-project-yocto-project.

[4] Evan-Amos.
public domain.
URL: https://commons.wikimedia.org/wiki/File:Linksys-Wireless-G-Router.jpg.

[5] Jwrodgers.
cc-by-sa-3.0.
URL: https://commons.wikimedia.org/wiki/File:RaspberryPi.jpg.

[6] André Karwath.
cc-by-sa-2.5.
URL: https://commons.wikimedia.org/wiki/File:Digital_Multimeter_Aka.jpg.

[7] Luikk.
cc-by-sa-3.0.
URL: https://commons.wikimedia.org/wiki/File:Development_Kit_EDK6446.jpg.

[8] M. Muruganandam.
URL: http://www.slideshare.net/murugan_m1/embedded-system-basics.

Literature II

Literatur & Bild-Nachweise

[9] OMAPpedia.
Bootloader project.
URL: http://omappedia.org/wiki/Bootloader_Project.

[10] randofo.
My useless machine.
URL: http://www.instructables.com/file/FNC31TMGL4Z6BE8/.

[11] Gary Richmond.
The large hadron collider switches on. if it's the end of the world, it will be powered by gnu/linux.
Free Software Magazine, 2008.
URL: http://www.freesoftwaremagazine.com/articles/large_hadron_collider_switches_if_its_end_world_it_will_be_powered_gnu_linux.

[12] Scout.
public domain.
URL: https://openclipart.org/detail/196174/question-girl.

[13] Scout.
public domain.
URL: https://openclipart.org/detail/191766/question-guy.

[14] Wikipedia.
Atlas experiment — wikipedia, the free encyclopedia, 2015.
[Online; accessed 11-October-2015].
URL: https://en.wikipedia.org/w/index.php?title=ATLAS_experiment&oldid=683613146.

[15] Wikipedia.
Eingebettetes system — wikipedia, die freie enzyklopädie, 2015.
[Online; Stand 10. Oktober 2015].
URL: https://de.wikipedia.org/w/index.php?title=Eingebettetes_System&oldid=145312335.

[16] Wikipedia.
Firmware — wikipedia, die freie enzyklopädie, 2015.
[Online; Stand 11. Oktober 2015].
URL: https://de.wikipedia.org/w/index.php?title=Firmware&oldid=146899876.