



a fool...



is getting cool...



Überblick

Grundlagen

- Motivation

- Theoretische Grundlagen

Optimierungen

- Bootloader

- Kernel

- Dateisystem

- Applikation

Beispiel

- Analyse des Testsystems

- Optimierung des Testsystems

Grundlagen

Motivation

Theoretische Grundlagen

Optimierungen

Bootloader

Kernel

Dateisystem

Applikation

Beispiel

Analyse des Testsystems

Optimierung des Testsystems

Motivation

- ▶ Automotiveanwendungen
- ▶ Energie sparen

Grundlagen

Motivation

Theoretische Grundlagen

Optimierungen

Bootloader

Kernel

Dateisystem

Applikation

Beispiel

Analyse des Testsystems

Optimierung des Testsystems

Der erste Schritt: Anforderungsdefinition

- ▶ Wo liegt die Obergrenze für die Bootzeit?
- ▶ Welche Funktionalität muß nach dieser Zeit zur Verfügung stehen?
- ▶ Geschwindigkeit vs. Flexibilität

Teilkomponenten des Bootprozesses



Teilkomponenten des Bootprozesses

- ▶ Hardware-Reset
- ▶ Bootloader
- ▶ Betriebssystem (Laden von Treibern, Einbinden des Dateisystems)
- ▶ Startskripte / Applikation

Kritische Hardwarekomponenten

- ▶ Netzteil
- ▶ Resetlogik
- ▶ Bootlogik / Bootreihenfolge
- ▶ Anbindung des Bootmediums
- ▶ Anbindung der benötigten Peripherie

Wichtig: Die Hardware ist wesentlicher Bestandteil eines Fastbootkonzeptes!!

Bootloader

- ▶ "Basiskonfiguration" der CPU
- ▶ Aufsetzen der sogenannten ATAGs Struktur
- ▶ Flushen der Caches
- ▶ Ausschalten der MMU

Der Linux Kernel

- ▶ Viele Funktionen zur Bootzeitoptimierung
- ▶ Sehr flexibel
- ▶ Kompression
- ▶ Möglichkeiten zur Parallelisierung von Initialisierungen
- ▶ Ca. 150ms - 250ms bis zum Einbinden des Dateisystems

Die Applikation

- ▶ In bestehenden Systemen oft am meisten Optimierungspotential
- ▶ Startskripte
- ▶ Linking

Bootprozeß analysieren



Grundlagen

- Motivation

- Theoretische Grundlagen

Optimierungen

- Bootloader

- Kernel

- Dateisystem

- Applikation

Beispiel

- Analyse des Testsystems

- Optimierung des Testsystems

Bootloader-Optimierungen (U-Boot) 1

Entfernen von im Feld nicht benötigten Features:

```
1 /* include/configs/boardname.h */
2 [...]
3 #include <config_cmd_default.h>
4 #undef CONFIG_CMD_NET
5 [...]
```

Bootloader-Optimierungen (U-Boot) 2

Verifizieren des Kernel Images:

```
setenv verify n
```

Keine Ausgaben der U-Boot Konsole:

```
setenv silent 1
```

Warten auf Nutzereingabe:

```
setenv bootdelay 0
```

Grundlagen

- Motivation

- Theoretische Grundlagen

Optimierungen

- Bootloader

- Kernel

- Dateisystem

- Applikation

Beispiel

- Analyse des Testsystems

- Optimierung des Testsystems

Kerneloptimierungen: Konfiguration

```
General setup --->
```

```
Kernel compression mode -->
```

- ▶ LZO für Embeddedsysteme sehr interessant
- ▶ Kopieren vs. Dekomprimieren
- ▶ Für Speichermedien mit wahlfreiem Lesezugriff steht auch "Execute in Place (XIP)" zur Verfügung

Kerneloptimierungen: Commandlineparameter

- ▶ Parameter zur Analyse der Laufzeit im Bootprozeß:
"initcall_debug", "printk_time=1"
- ▶ Delay Loop Calibration: "lpj="; Kann auf ARM9 Systemen
> 100ms einsparen.

Kerneloptimierungen: Delayloop

```
...  
[0.018847] Calibrating delay loop...  
           626.68 BogoMIPS (lpj=3133440)  
[0.316717] pid_max: default: 32768 minimum: 301  
...
```

Kerneloptimierungen: initcall_debug

```
[0.452115] calling exceptions_init+0x0/0x90 @ 1
[0.452172] initcall exceptions_init+0x0/0x90
           returned 0 after 0 usecs
[0.452203] calling versatile_i2c_init+0x0/0x24 @ 1
[0.452321] initcall versatile_i2c_init+0x0/0x24
           returned 0 after 0 usecs
[0.452352] calling pl011_init+0x0/0x54 @ 1
[0.452382] Serial: AMBA PL011 UART driver
[0.453647] dev:f1: ttyAMA0 at MMIO 0x101f1000
           (irq = 12) is a PL011 rev1
[0.481540] console [ttyAMA0] enabled
...
[0.484427] initcall pl011_init+0x0/0x54
           returned 0 after 29296 usecs
```

bootgraph.pl



bootgraph.pl

1. Vorbereitung: Booten mit "initcall__debug loglevel=8

2. Auf dem Target:

```
$ dmesg > bootlog.txt
```

3. Auf dem Host:

```
$ cd linux-XXX
```

```
$ cat /path_to_rfs/bootlog.txt | \  
perl scripts/bootgraph.pl > bootlog.svg
```

scripts/bootgraph.pl

0.42



0.45



0.47



Grundlagen

Motivation

Theoretische Grundlagen

Optimierungen

Bootloader

Kernel

Dateisystem

Applikation

Beispiel

Analyse des Testsystems

Optimierung des Testsystems

InitRAMFS: Mehrstufiges booten

Initprozeß für das InitRAMFS wird mit `rdinit=` festgelegt. Zum Beispiel: `rdinit=/etc/init.d/start.sh`

InitRAMFS: Mehrstufiges booten

```
/etc/init.d/start.sh:
```

```
#!/bin/sh
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devtmpfs devtmpfs /dev

mount /dev/mmcblk0p1 /media
fb splash -s /media/splash.ppm -d /dev/fb0

mount -o move /proc /mnt/proc
mount -o move /sys /mnt/sys
mount -o move /dev /mnt/dev

exec switch_root /mnt /linuxrc
```

Grundlagen

- Motivation

- Theoretische Grundlagen

Optimierungen

- Bootloader

- Kernel

- Dateisystem

- Applikation

Beispiel

- Analyse des Testsystems

- Optimierung des Testsystems

Optimierungen der Applikation

- ▶ Analyse des Startprozesses mit bootchartd
- ▶ Ggf. direktes Starten der Applikation mit init=
- ▶ Überprüfung dynamisch gelinkter Applikationen

Dynamisches Linken

1. DT_RPATH Sektion des ELF Executables
2. Pfade, die in LD_LIBRARY_PATH spezifiziert sind
3. DT_RUNPATH Sektion des ELF Executables
4. Binärfile /etc/ld.so.cache
5. Defaultpfade /lib und /usr/lib

Dynamisches Linken: Analyse

```
$ LD_DEBUG=libs ls
3082: find library=librt.so.1 [0];
      searching
3082: search cache=/etc/ld.so.cache
3082: trying file=/lib/librt.so.1
```

Grundlagen

- Motivation

- Theoretische Grundlagen

Optimierungen

- Bootloader

- Kernel

- Dateisystem

- Applikation

Beispiel

- Analyse des Testsystems

- Optimierung des Testsystems

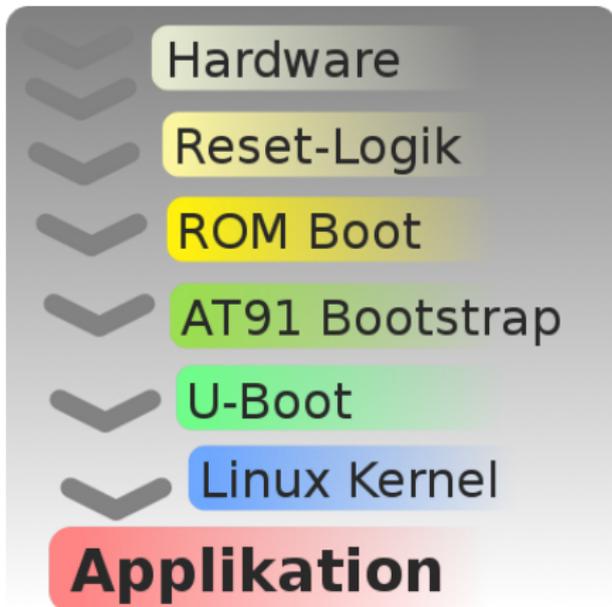
Testsystem

- ▶ ARM9 CPU der Atmel AT91 Serie
- ▶ Ausgangspunkt: Busybox Image der Angstrom Distribution
- ▶ Speichermedium: NAND-Flash
- ▶ Testapplikation: Toggeln eines GPIO über das SysFS GPIO-Interface

Bootmodi der AT91 Familie

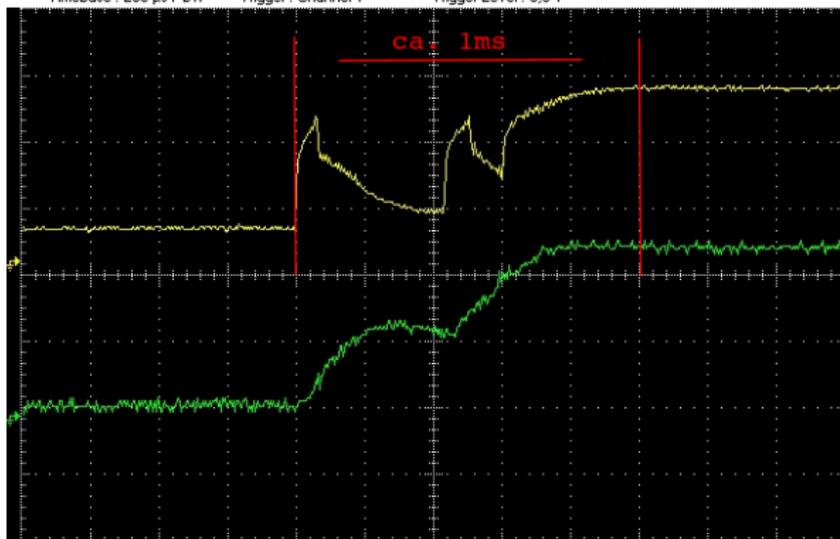
- ▶ RomBOOT: Booten über eine interne Bootlogik
- ▶ Booten über CS0 des External Bus Interface

AT91 RomBOOT



Einschaltverhalten / Netzteil

Wittig Technologies AG - W2000 Series - 09.09.2010 16:11:42 - QP_Scope01
Timebase : 200 μ s / Div Trigger : Channel 1 Trigger Level : 6,3V



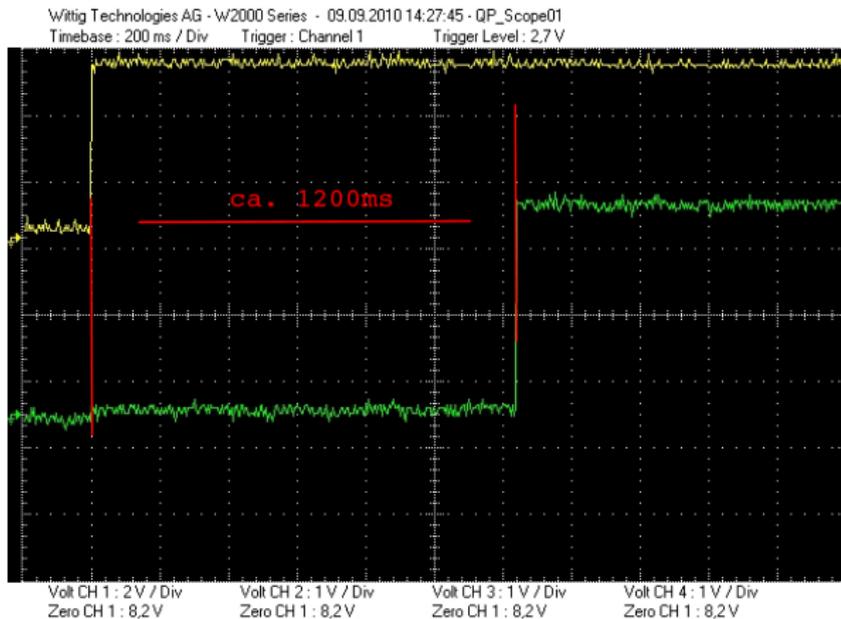
Volt CH 1 : 5V / Div
Zero CH 1 : 25,2V

Volt CH 2 : 2V / Div
Zero CH 1 : 25,2V

Volt CH 3 : 1V / Div
Zero CH 1 : 25,2V

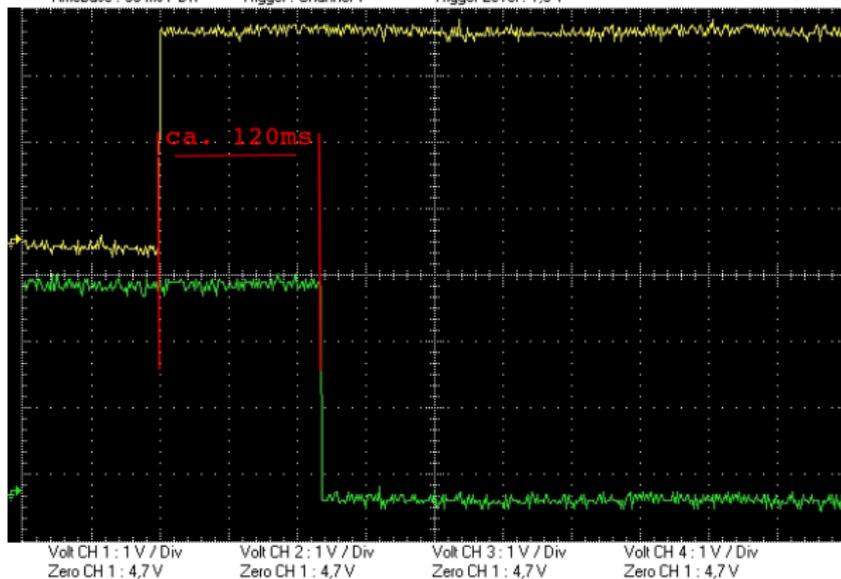
Volt CH 4 : 1V / Div
Zero CH 1 : 25,2V

Resetverhalten



RomBOOT

Wittig Technologies AG - W2000 Series - 09.09.2010 14:37:24 - QP_Scope01
Timebase : 50 ms / Div Trigger : Channel 1 Trigger Level : 1,3V



Fazit / Hardwareoptimierungen

- ▶ Internen Oszillator für Slowclock verwenden: > 1s Ersparnis
- ▶ Booten von CS0: 100ms - 150ms Ersparnis

Grundlagen

Motivation

Theoretische Grundlagen

Optimierungen

Bootloader

Kernel

Dateisystem

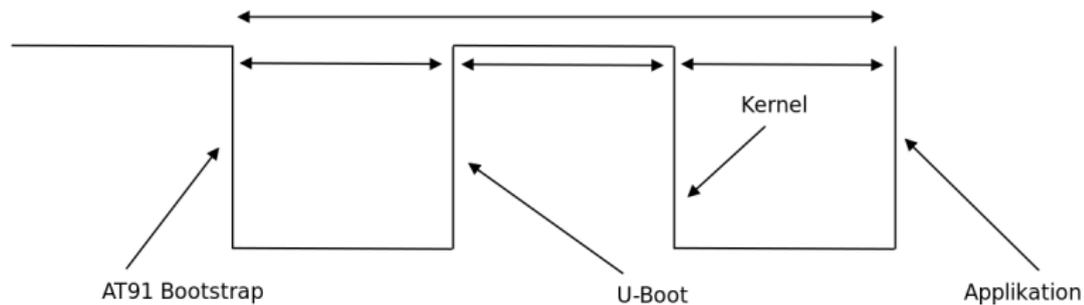
Applikation

Beispiel

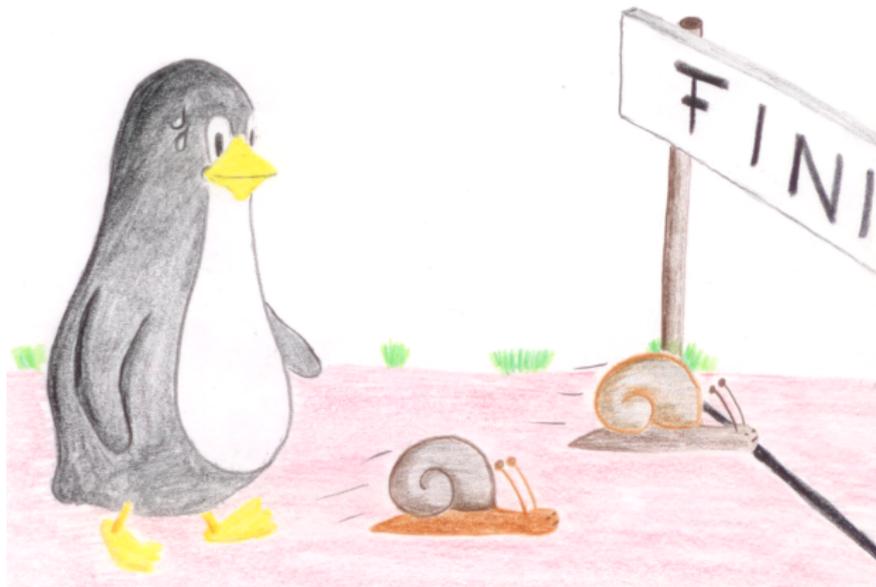
Analyse des Testsystems

Optimierung des Testsystems

Bootzeitmessung mittels GPIO



Initiales Bootverhalten



Initiales Bootverhalten

Messpunkt	Zeit
Bootstrap - Uboot	—
Uboot - Kernel	6,5s
Kernel - Applikation	4,5s
Gesamt	11s

Einfache Optimierungen



U-Boot ohne Netzwerkunterstützung

Messpunkt	Zeit
Bootstrap - Uboot	—
Uboot - Kernel	4,25s
Kernel - Applikation	4,5s
Gesamt	8,75s

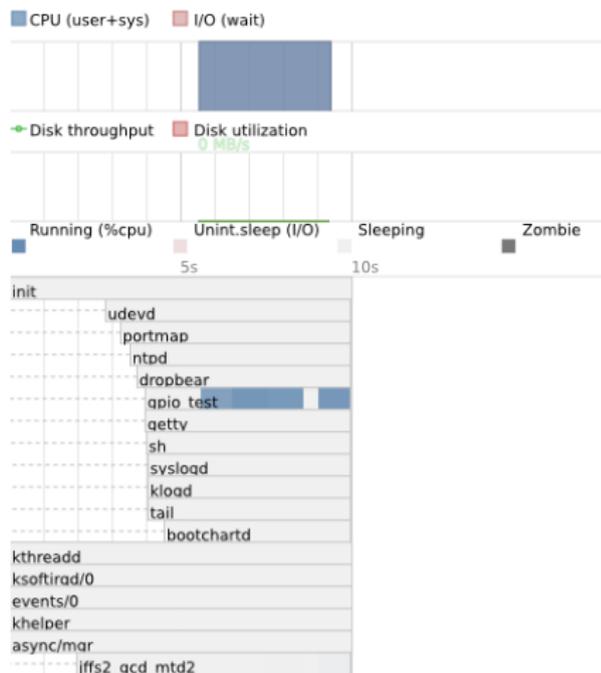
U-Boot verify=n

Messpunkt	Zeit
Bootstrap - Uboot	—
Uboot - Kernel	3,89s
Kernel - Applikation	4,5s
Gesamt	8,39s

Kernel "abspecken"

Messpunkt	Zeit
Bootstrap - Uboot	—
Uboot - Kernel	3,77s
Kernel - Applikation	4,33s
Gesamt	8,1s

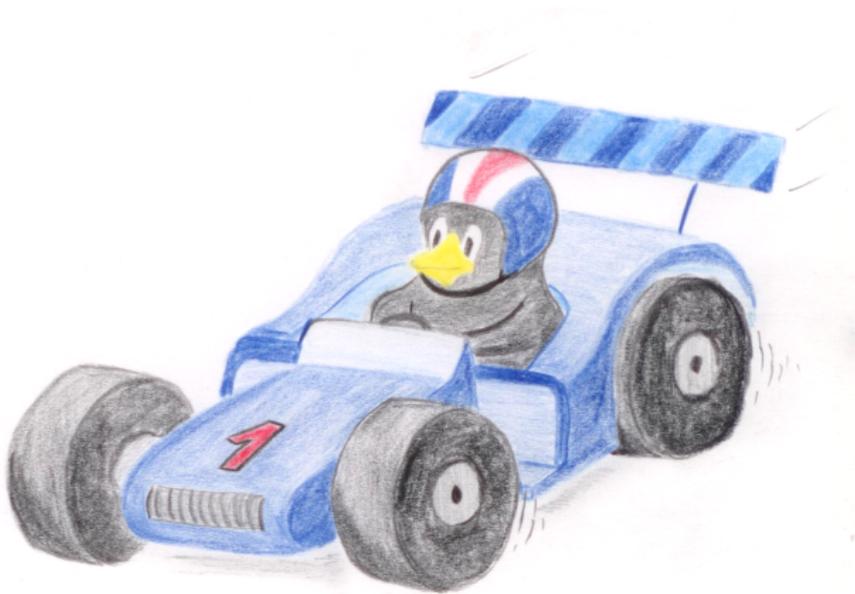
Analyse der Startskripte: Bootchartd



Optimieren der Startskripte

Messpunkt	Zeit
Bootstrap - Uboot	—
Uboot - Kernel	3,77s
Kernel - Applikation	3,61
Gesamt	7,38s

Booten über eine RAMdisk / InitRAMFS

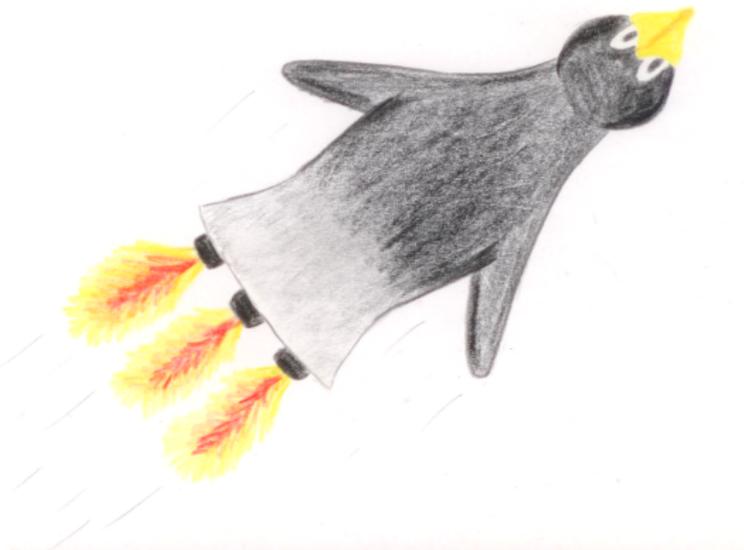


LZO komprimiertes InitRAMFS

Die Applikation wird direkt mit `init=` angestartet.

Messpunkt	Zeit
Bootstrap - Uboot	—
Uboot - Kernel	3,790s
Kernel - Applikation	0,584s
Gesamt	4,374s

Modifizierter AT91 Bootstrap



Modifizierter AT91 Bootstrap

AT91 Bootstrap startet direkt Linux an. D.h. es wird ohne U-Boot gebootet.

Messpunkt	Zeit
Bootstrap - Kernel	676ms
Kernel - Applikation	584ms
Gesamt	1,260s

lpj=

Messpunkt	Zeit
Bootstrap - Kernel	676ms
Kernel - Applikation	384ms
Gesamt	1,060s

< 1s !!



Keine Ausgaben auf serielle Schnittstelle (quiet)

Messpunkt	Zeit
Bootstrap - Kernel	524ms
Kernel - Applikation	212ms
Gesamt	736ms

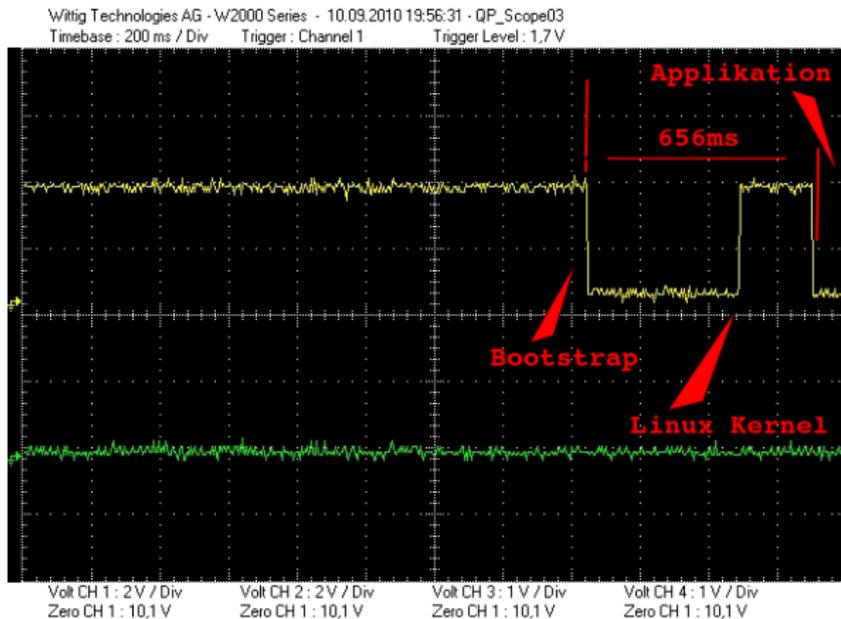
Zieleinlauf ;-)



LZO komprimiertes Kernelimage

Messpunkt	Zeit
Bootstrap - Kernel	444ms
Kernel - Applikation	212ms
Gesamt	656ms

Bootverhalten nach den Optimierungen



Fazit

- ▶ Die Hardware ist wesentlicher Bestandteil eines Fastbootkonzeptes
- ▶ Linux bietet die optimale Plattform für jeden, der ein modernes Betriebssystem verwenden möchte, aber in weniger als 1s booten muß
- ▶ Bereits mit einfachen Optimierungen können mehrere Sekunden gespart werden

That's it

