

Reducing the boot time of Linux devices

Chris Simmonds

foss-north 2019



License



These slides are available under a Creative Commons Attribution-ShareAlike 4.0 license. You can read the full text of the license [here](http://creativecommons.org/licenses/by-sa/4.0/legalcode)

<http://creativecommons.org/licenses/by-sa/4.0/legalcode>

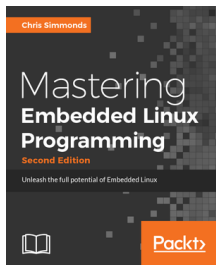
You are free to

- copy, distribute, display, and perform the work
- make derivative works
- make commercial use of the work

Under the following conditions

- Attribution: you must give the original author credit
- Share Alike: if you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one (i.e. include this page exactly as it is)
- For any reuse or distribution, you must make clear to others the license terms of this work

About Chris Simmonds



- Consultant and trainer
- Author of *Mastering Embedded Linux Programming*
- Working with embedded Linux since 1999
- Android since 2009
- Speaker at many conferences and workshops

"Looking after the Inner Penguin" blog at <http://2net.co.uk/>



@2net_software



<https://uk.linkedin.com/in/chrisdsimmonds/>

Overview

- Shorter boot time is always desirable

Overview

- Shorter boot time is always desirable
- It is always possible to reduce boot time

Overview

- Shorter boot time is always desirable
- It is always possible to reduce boot time
 - This is software: there is always more you can do

Overview

- Shorter boot time is always desirable
- It is always possible to reduce boot time
 - This is software: there is always more you can do
- So, the real issues are:

Overview

- Shorter boot time is always desirable
- It is always possible to reduce boot time
 - This is software: there is always more you can do
- So, the real issues are:
 - How much effort do you want to put in?

Overview

- Shorter boot time is always desirable
- It is always possible to reduce boot time
 - This is software: there is always more you can do
- So, the real issues are:
 - How much effort do you want to put in?
 - How big a mess will you leave when you are done?

Where it all goes wrong

- The more effort you put in to reducing boot time, the higher the future maintenance burden



Before

Where it all goes wrong

- The more effort you put in to reducing boot time, the higher the future maintenance burden



Before

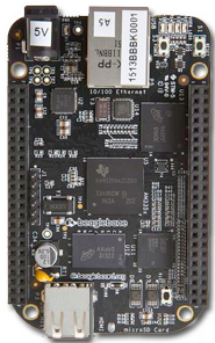


After

Getting it right

- This is a **pragmatic** guide to reducing boot time
- Using the three guiding principles
 - Measure
 - Evaluate
 - Modify
- With inspiration from Andrew Murray's presentation "12 Lessons learnt in boot time reduction"

Example system



- BeagleBone Black (single core Cortex A-8, 1GHz)
- Built using Yocto Project Pyro
- Qt app (qt4e-demo-image)
- systemd init

Measuring boot time

Commonly used tools:

- **Grabserial**: adds timestamps to all serial console output
- **Bootchart**: graphs running time of user-space processes
- **Bootgraph**: graphs running time of kernel functions

You can also change outputs (GPIOs) at certain points in the code and use an **oscilloscope** to monitor them. This is the only way you can measure the time from *power-on*

Grabserial

- Get it from <https://elinux.org/Grabserial> (written by Tim Bird)
- You will need to install python-serial as well

```
sudo apt install python-serial
```

Then:

```
grabserial -d /dev/ttyUSB0 -t -m "U-Boot SPL" -e 30 -o boot.log
```

-d = serial device

-t = print timestamps

-m = match string before starting to log

-e = number of seconds to capture for

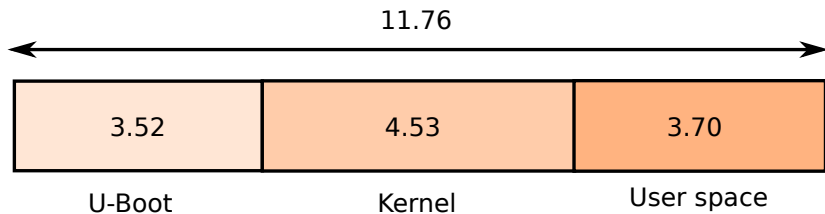
(useful when redirecting to a file: makes sure file is closed cleanly
and all output is recorded)

-o [filename] = output to file

Grabserial output

```
[0.000000 0.000000]
[0.000398 0.000398] U-Boot SPL 2017.01 (Oct 17 2017 - 11:10:21)
[0.095607 0.095607] Trying to boot from MMC1MMC partition switch failed
[0.188577 0.092970] *** Warning - MMC partition switch failed, using default
[0.190962 0.002385]
[0.203591 0.012629] reading u-boot.img
[0.204300 0.000709] reading u-boot.img
[0.259620 0.055320]
[0.259727 0.000107]
[0.259781 0.000054] U-Boot 2017.01 (Oct 17 2017 - 11:10:21 +0100)
[0.261334 0.001553]
[0.261384 0.000050] CPU   : AM335X-GP rev 2.0
[...]
11.756176 0.080002] Started qtdemo.
```


Baseline measurement



From power-on to starting qtdemo app, in seconds

Reducing user-space boot time

- User-space is often simplest to change, so we start here
- Typical modifications
 - **Easy**: running order in init
 - **Quite easy**: build optimizations, e.g. compile flags
 - **Difficult**: library optimizations to reduce load time

Measuring init boot time using Bootchart

- Two parts
 - **bootchartd** captures process stats, starting with init
 - part of Busybox, and also a package in Yocto Project/OpenEmbedded
 - **bootchart** analyses offline, and creates a graph
- <http://www.bootchart.org>
- Launch it in place of init; it will fork and then exec `/sbin/init`

```
init=/sbin/bootchartd
```

- Logs are put in `/var/log/bootchart.tgz`
- Example on next slide: you are not expected to be able to read it!

Bootchart

Boot chart for beaglebone.(none) (Thu Oct 19 07:59:37 2017)

uname: Linux 4.10.17-yocto-standard #1 PREEMPT Tue Oct 17 11:20:08 BST 2017 armv7l

release: Poky (Yocto Project Reference Distro) 2.3.2

CPU: ARMv7 Processor rev 2 (v7l) 1

kernel options: console=ttyO0,115200n8 root=PARTUUID=b6ce20d3-02 rw rootfstype=ext4 rootwait init=/sbin/bootchartd

time: 00:38.15

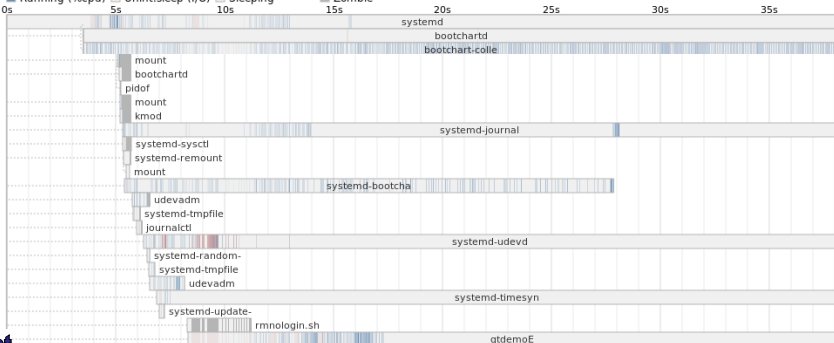
■ CPU (user+sys) ■ I/O (wait)



↗ Disk throughput ■ Disk utilization



■ Running (%cpu) ■ Unint.sleep (I/O) ■ Sleeping ■ Zombie



Optimizing user space

- Careful analysis of bootchart shows that qtdemo is not started until 3.5 seconds after init begins
- Since this is the most important thing to run, we need to move it up the booting order
- Some options:
 - If using System V init, use a lower number for the "S" script, e.g. "S01"
 - If using systemd, change the dependencies of the unit (example at end of slides)
 - Or, run qtdemo ahead of init (*)

(*) be aware that the rootfs will most likely be ro, that other fs will not be mounted, and that the network will not be started

Running the app as init

- This Script runs qtdemo in the background and then execs the normal init so that it is still PID 1:

```
/usr/bin/run-qtdemo.sh
```

```
#!/bin/sh
```

```
/usr/bin/qtdemoE -qws &
```

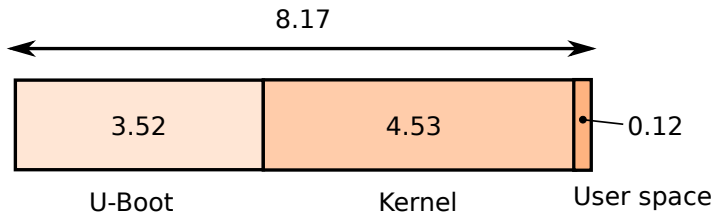
```
sleep 1
```

```
exec /sbin/init
```

- Change kernel command line:

```
init=/usr/bin/run-qtdemo.sh
```

First pass



Launching qtdemo ahead of init saves **3.5 s**

Reducing kernel boot time

- Typical modifications
 - **Easy**: reduce console messages by adding `quiet` to cmdline
 - **Moderate**: slim down kernel by removing drivers, filesystems, subsystems
 - **Moderate**: slim down device tree by removing hardware interfaces that are not used
 - **Tricky**: set about optimizing device probe functions

Measuring kernel boot time: Bootgraph

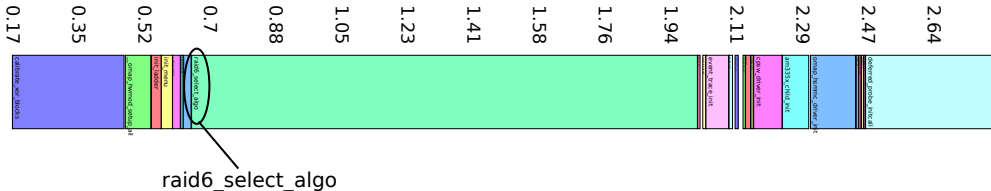
- **Bootgraph** analyses kernel functions during boot
- Configure kernel with `CONFIG_PRINTK_TIME` and `CONFIG_KALLSYMS`
- Add `initcall_debug` to kernel cmdline
- Then boot, and get the kernel log

```
dmesg > boot.log
```

- Back on the host, create the graph using

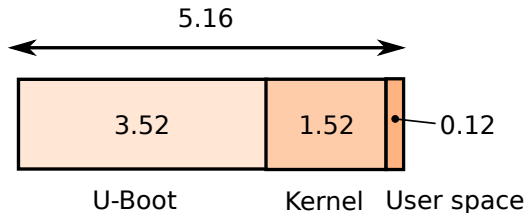
```
linux/scripts/bootgraph.pl boot.log > boot.svg
```

Bootgraph



- Note: almost 2 seconds in `raid6_select_algo`
- RAID6 is selected by `CONFIG_BTRFS_FS`, which we are not using. So, rip it out!

Second pass



- Remove BTRFS: saves 1860 ms
- Add "quiet" to cmdline: saves 700 ms
- Cut out unused drivers (e.g. WiFi), reducing zImage from 5.6 MiB to 3.2 MiB: saves 450 ms
- Total saving: **3.01 s**

Reducing bootloader boot time

- Typical modifications
 - **Easy**: remove the boot delay
 - **Easy(ish)**: simplify boot scripts
 - **Medium**: compile out functions you don't need
 - **Harder**: use falcon mode

Remove boot delay

- U-Boot usually configured with a delay, during which you can press space bar to stop booting

```
Press SPACE to abort autoboot in 2 seconds
```

- Set the display to zero

```
setenv bootdelay 0
```

- You can still hit space bar to halt boot *if you are quick enough*

OK, this is a bit trivial, but it still needs to be done

Simplify boot scripts 1/3

- Bootscript for BeagleBone is c. 100 lines
 - Testing all possible sources of boot images takes time
 - Loading image from filesystems takes time
- Reduce the script to the cases you support
- Loading from raw MMC sectors is fastest

Simplify boot scripts 2/3

Reduce this (lines 22 to 122 omitted)...

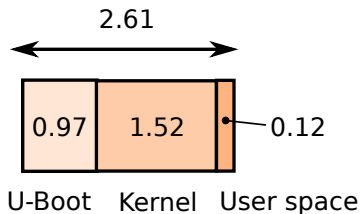
```
arch=arm
args_mmc=run finduuid;setenv bootargs console=${console} ${optargs} root=PARTUUID=${uuid} rw rootfstype=${mmccrootfstype}
baudrate=115200
board=am335x
board_name=A335BNLT
board_rev=0A5C
board_serial=2413BBBK1972
boot_a_script=load ${devtype} ${devnum}:${distro_bootpart} ${scriptaddr} ${prefix}${script}; source ${scriptaddr}
boot_efi_binary=if fdt addr ${fdt_addr_r}; then bootefi bootmgr ${fdt_addr_r};else bootefi bootmgr ${fdtcontroladdr};fi;load ${devtype}
if fdt addr ${fdt_addr_r}; then bootefi ${kernel_addr_r} ${fdt_addr_r};else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi
boot_extlinux=sysboot ${devtype} ${devnum}:${distro_bootpart} any ${scriptaddr} ${prefix}extlinux/extlinux.conf
boot_fdt=try
boot_fit=0
boot_net_usb_start=usb start
boot_prefixes=/ /boot/
boot_script_dhcp=boot.scr.uimg
boot_scripts=boot.scr.uimg boot.scr
boot_targets=mmc0 legacy_mmc0 mmc1 legacy_mmc1 nand0 pxe dhcp
bootcmd=if test ${boot_fit} -eq 1; then run update_to_fit; fi; run findfdt; run init_console; run envboot; run distro_bootcmd
bootcmd_dhcp=run boot_net_usb_start; if dhcp ${scriptaddr} ${boot_script_dhcp}; then source ${scriptaddr}; fi;setenv efi_fdtfile ${fdt
${soc}-${board}${boardver}.dtb; fi; setenv efi_old_vci ${bootp_vci};setenv efi_old_arch ${bootp_arch};setenv bootp_vci PXEClient:Arch
n tftpboot ${fdt_addr_r} dtb/${efi_fdtfile};if fdt addr ${fdt_addr_r}; then bootefi ${kernel_addr_r} ${fdt_addr_r}; else bootefi ${ker
tenv bootp_arch ${efi_old_arch};setenv efi_fdtfile;setenv efi_old_arch;setenv efi_old_vci;
bootcmd_legacy_mmc0=setenv mmcdev 0; setenv bootpart 0:2 ; run mmcboot
bootcmd_legacy_mmc1=setenv mmcdev 1; setenv bootpart 1:2 ; run mmcboot
bootcmd_mmc0=setenv devnum 0; run mmc_boot
bootcmd_mmc1=setenv devnum 1; run mmc_boot
[...]
```

Simplify boot scripts 3/3

To this...

```
setenv bootargs console=tty00 root=/dev/mmcblk0p2 quiet
fatload mmc 0:1 82000000 zImage
fatload mmc 0:1 83000000 am335x-boneblack.dtb
bootz 82000000 - 83000000
```


Third pass



- Remove boot delay: saves 2000 ms
- Simplify boot script: saves: 230 ms
- Total saving: **2.32 s**

Conclusion

- Boot time reduction need not require massive changes to the platform code
- Bake the changes you do make into the build system
- Look to the future

Slides

[www.slideshare.net/chrissimmonds/
reducing-the-boot-time-of-linux-devices](http://www.slideshare.net/chrissimmonds/reducing-the-boot-time-of-linux-devices)

This and other topics associated with using embedded Linux are covered in my training courses

Full details are available on the 2net web site
<http://www.2net.co.uk/training.html>

Contact <mailto:training@2net.co.uk> if you have a question or you would like to book a course

Systemd unit

- Unit for a "normal" systemd service that is started by multi-user target

```
/lib/systemd/system/qtdemo.service
```

```
[Unit]
Description=qtdemo

[Service]
ExecStart=/usr/bin/qtdemoE -qws

[Install]
WantedBy=multi-user.target
```

Systemd unit

- **sysinit** target is started *really early*
 - Also, we disable default dependencies

```
[Unit]
Description=qtdemo
DefaultDependencies=no

[Service]
ExecStart=/usr/bin/qtdemoE -qws

[Install]
WantedBy=sysinit.target
```

See man entry for `systemd.special(7)` for more information about targets

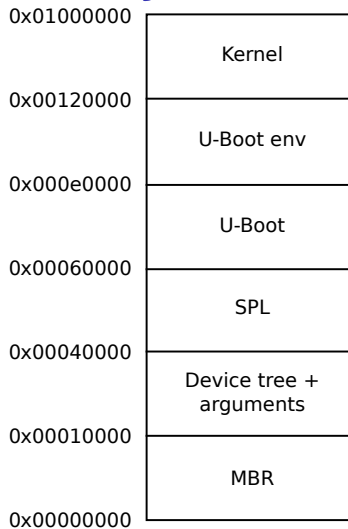
Falcon mode

- Most SoCs have three stages of bootloader
 - 1: on-chip ROMcode, which loads SPL into static RAM
 - 2: SPL, which has to be small enough to fit in SRAM. Main job is to set up DRAM and load bootloader into it
 - 3: bootloader, which loads the kernel (4th stage)
- U-Boot Falcon mode jams the SPL and bootloader into the SRAM
- No user interaction: uses pre-built environment block

U-Boot Falcon mode

- See `doc/README.falcon` in U-Boot source code
- Steps
 - Configure U-Boot
 - Copy to SD card
 - Boot into U-Boot and prepare environment
 - Save environment in MMC

MMC layout



-
- Layout of the first 16MiB (128K sectors) on BeagleBone **without** Falcon mode
 - Other SoCs are similar

Configure U-Boot

- Tell U-Boot where to put things
- Add to `include/configs/am335x_evm.h`:

```
#define CONFIG_SYS_MMC_ENV_DEV      0
#define CONFIG_ENV_OFFSET           0xe0000
```

- Build U-Boot in the normal way

```
$ export CROSS_COMPILE=arm-buildroot-linux-gnueabi-
$ make am335x_boneblack_defconfig
$ make
```

Copy images to micro SD card

- Using the layout from the diagram shown earlier
- Note: requires a uImage, not zImage

```
$ sudo dd if=am335x-boneblack.dtb of=/dev/mmcblk0 bs=1 seek=65536
$ sudo dd if=ML0 of=/dev/mmcblk0 bs=1 seek=262144
$ sudo dd if=u-boot.img of=/dev/mmcblk0 bs=1 seek=393216
$ sudo dd if=uImage of=/dev/mmcblk0 bs=1 seek=1179648
```

Prepare Falcon boot 1/2

```
=> mmc read 82000000 900 2000
=> mmc read 83000000 80 180
=> spl export fdt 82000000 - 83000000
## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   Linux-4.10.17-yocto-standard
   Created:      2017-10-21 13:12:02 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3365704 Bytes = 3.2 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 83000000
   Booting using the fdt blob at 0x83000000
   Loading Kernel Image ... OK
   Loading Device Tree to 8fff4000, end 8ffffc86 ... OK
   Loading Device Tree to 8ffe5000, end 8fff3c86 ... OK
Argument image is now in RAM: 0x8ffe5000
```

Prepare Falcon boot 2/2

- Write the modified FDT and parameter block back to MMC:

```
=> mmc write 0x8ffe5000 80 180
```

- Enable falcon mode:

```
=> setenv boot_os 1
```

- Now re boot and it should jump straight from SPL to kernel:

```
U-Boot SPL 2017.05-rc3-dirty (Oct 21 2017 - 11:27:01)
Trying to boot from MMC1
Booting Linux on physical CPU 0x0
Linux version 4.10.17-yocto-standard (oe-user@oe-host)
```