Fastboot Tools and Techniques

mentor embedded

mentor.com/embedded

Android is a trademark of Google Inc. Use of this trademark is subject to Google Permissions.
Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.
Qt is a registered trade mark of Digia Plc and/or its subsidiaries. All other trademarks mentioned in this document are trademarks of their respective owners.

Agenda

- Definitions
- Hardware/Software Architecture
- Typical Embedded Linux System
  - Boot Sequence
- Measurement Methods
- Optimization Techniques

What is Fast Boot

- Fast Boot is not a single technology or product
- Many techniques are architecture or platform dependent
- Product/application defines fast boot requirements
- Your system architecture can determine limits
  - Splash/sound in 1 second
  - Camera video in 2 seconds
  - Partial HMI in 3 seconds
  - Full multimedia plus networking in 4 seconds
  - Secure Boot?

What takes so much time?

- Power/Clock Stabilization
  - usually negligible but s.b. considered
- Low Level CPU Initialization - ~ 100 ms
  - Bootloader (often multi-stage, ie secure boot)
- Loading images (kernel, u-boot, rootfs, dtb)
  - Even a very small kernel can be 1-2 MB (compressed)
  - Multiple DMA transfers
- Subsystem (Driver) initialization
- Mounting a root file system
- Init – System Utilities and Applications

Typical Embedded Linux System

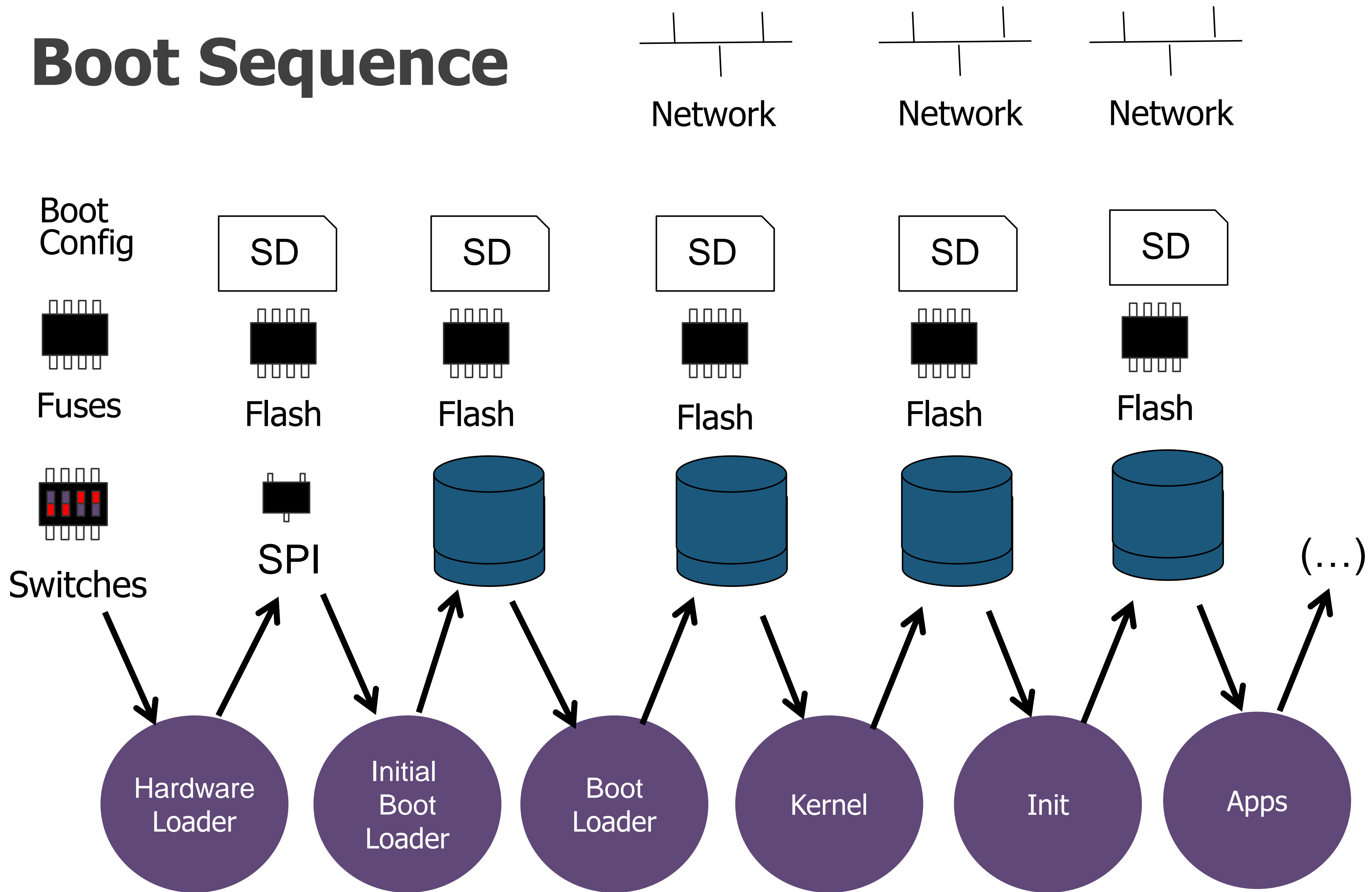
- Freescale i.MX6 SabreLite:
  - Quad-Core ARM® Cortex A9 1 GHz
  - Yocto core-image-sato
- Stopwatch analysis:

From Power ON to:	Time (seconds)
Kernel FB logo	5
Userland psplash	13
Full Mobile Desktop	23

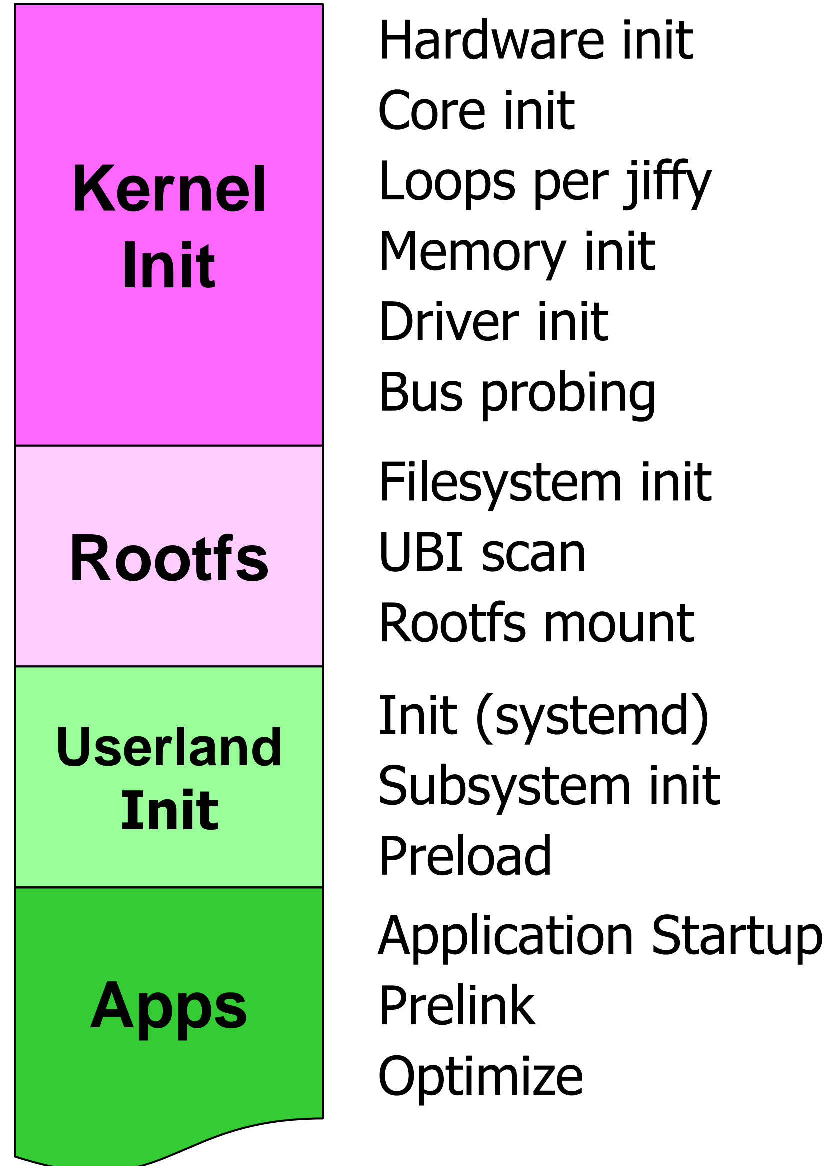
Hardware Considerations

- Hardware architecture makes a difference
  - Goes beyond just clock speeds, etc.
- Power and Clock stabilization can be very fast
- Design choices should support fast boot requirements
- Examples:
  - Loading u-boot and kernel from SPI NOR takes substantially longer than from parallel NOR
  - NAND flash or SD/MMC requires early software overhead but may be faster overall
  - enable caches early (in bootloader)
  - Class10 SD card boots in 6S vs 16S for std HC card

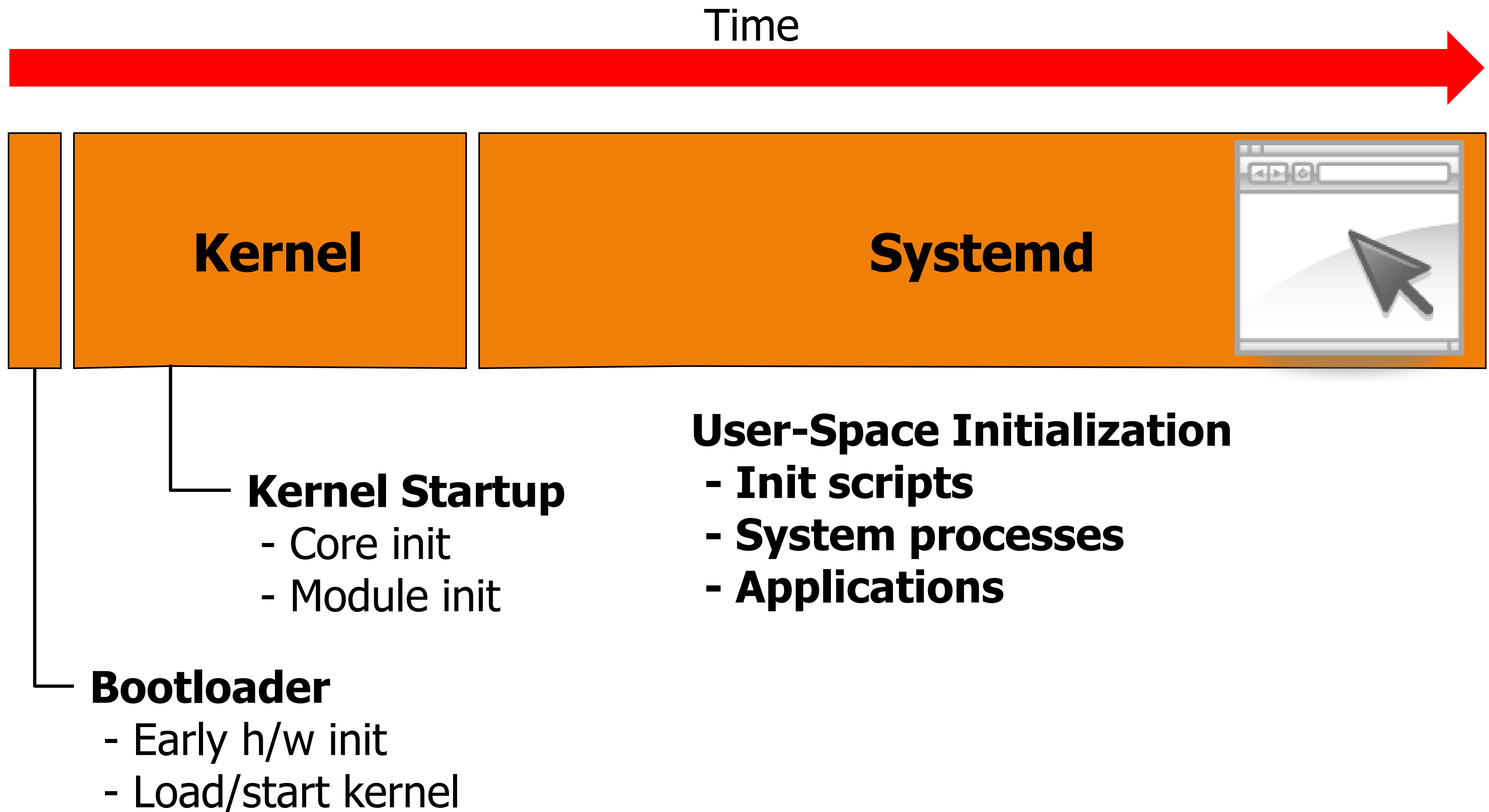
Boot Sequence



Typical Linux Boot



Relative Phase Lengths



Splash, Sounds and Progress Bars

- Indicates system is active, but still booting
- A splash can take place
  - In the bootloader
    - Gets splash up sooner, but...
      - Usually require some porting to bootloader
  - In the kernel
    - After initialization of the framebuffer driver
  - Early user-space init (psplash)
    - Before system apps initialized
- Do splash off the fastpath, if possible
  - May delay other functionality

MEASUREMENT

“Measurement is the first step that leads to control and eventually to improvement. If you can’t measure something, you can’t understand it. If you can’t understand it, you can’t control it. If you can’t control it, you can’t improve it.”
— H. James Harrington

Or, to put it more simply:

You cannot control what you do not measure

Some Popular Measurement Tools

- Ftrace
- Boot chart (userspace)
- SystemTap
- LTTng (kernel/userspace)
- Grabserial
- CONFIG_PRINTK_TIME
- initcall_debug
- Uptime (userspace)
- Oprofile (both)
- Strace (userspace)

Boot Time Measurement Methods

- Bootloader profiling (may require custom port)
- Several tools are helpful for Linux kernel profiling
  - Grabserial
  - CONFIG_PRINTK_TIME
  - Ftrace
  - LTTng
- Userland tools
  - Bootchart
  - LTTng
  - perf
  - Mentor Embedded System Analyzer

Profiling U-Boot

- Grabserial
- Ftrace
- U-boot logbuffer

Grabserial

```
$ sudo /tftpdir/grabserial -d /dev/ttyUSB0 -f -b 115200 -v -t -l -q login -c reset | tee
grabs_uboot_kernel.log
setting basetime to time of program launch Opening serial port /dev/ttyUSB0
115200:8N1:xonxoff=0:rtscts=0
Printing timing information for each line Instant pattern 'login' to exit program
[0.018431 0.018431] reset
[0.019071 0.000640] resetting ...
[0.348297 0.329226]
[0.348478 0.000181]
[0.348544 0.000066] U-Boot 2014.07 (Mar 13 2015 - 10:15:26)
[0.351763 0.003219]
[0.352010 0.000247] CPU:   Freescale i.MX6Q rev1.2 at 792 MHz
[0.355329 0.003319] Reset cause: WDOG
[0.356614 0.001285] Board: SABRE Lite
[0.358084 0.001470] I2C:   ready
[0.359014 0.000930] DRAM:  1 GiB
[0.360588 0.001574] Resetting the log with v3 settings
[0.381330 0.020742] MMC:   FSL_SDHC: 0, FSL_SDHC: 1
[0.478000 0.096670] auto-detected panel Hannstar-XGA
[0.487408 0.009408] Display: Hannstar-XGA (1024x768)
[0.518085 0.030677] initializing logbuff driverIn:    serial
[0.531724 0.013639] Out:   serial
[0.534808 0.003084] Err:   serial
[0.538137 0.003329] Net:   No ethernet found.
```

Grabserial (Cont'd)

```
[0.559949 0.021812] Warning: Your board does not use generic board. Please read
[0.583152 0.023203] doc/README.generic-board and take action. Boards not
[0.611813 0.028661] upgraded by the late 2014 may break or be removed.
[0.639403 0.027590] Hit any key to stop autoboot:  0
[0.655833 0.016430] switch to partitions #0, OK
[0.669308 0.013475] mmc1 is current device
[0.685046 0.015738]
[0.685379 0.000333] MMC read: dev # 1, block # 2048, count 12288 ... 12288 blocks read: OK
[0.947171 0.261792]
[0.947622 0.000451] MMC read: dev # 1, block # 1536, count 256 ... 256 blocks read: OK
[0.982004 0.034382] ## Booting kernel from Legacy Image at 12000000 ...
[0.995392 0.013388]    Image Name:   Linux-3.10.61-mel-fsl-imx6
[1.006737 0.011345]    Image Type:   ARM Linux Kernel Image (uncompressed)
[1.029166 0.022429]    Data Size:    5956720 Bytes = 5.7 MiB
[1.043012 0.013846]    Load Address: 10008000
[1.057490 0.014478]    Entry Point:  10008000
[1.072611 0.015121]    Verifying Checksum ... OK
[1.086180 0.013569] ## Flattened Device Tree blob at 18000000
[1.098066 0.011886]    Booting using the fdt blob at 0x18000000
[1.109720 0.011654]    Loading Kernel Image ... OK
[1.121417 0.011697]    Using Device Tree in place at 18000000, end 1800f8d7
[1.174149 0.052732]
[1.175402 0.001253] Starting kernel ...
[1.180844 0.005442]
[2.755653 1.574809] [    0.000000] Booting Linux on physical CPU 0x0
```

u-boot ftrace patches

- Original infrastructure by Simon Glass for x86/generic
  - Relatively easy to add an arch, e.g arm:

```
diff --git a/arch/arm/lib/board.c b/arch/arm/lib/board.c
@@ -229,4 +234,5 @@ static int mark_bootstage(void)
 init_fnc_t *init_sequence[] = {
+        trace_early_init,
         arch_cpu_init,          /* basic arch cpu dependent setup */
@@ -365,4 +371,11 @@ void board_init_f(ulong bootflag)

+#ifdef CONFIG_TRACE
+         addr -= CONFIG_TRACE_BUFFER_SIZE;
+         gd->trace_buff = (void *)addr;
+         debug("Reserving %dk for trace data at: %08lx\n",
+               CONFIG_TRACE_BUFFER_SIZE >> 10, addr);
+#endif
+
        /*
@@ -522,4 +535,8 @@ void board_init_r(gd_t *id, ulong dest_addr)

+#ifdef CONFIG_TRACE
+         trace_init(gd->trace_buff, CONFIG_TRACE_BUFFER_SIZE);
+#endif
+
```

u-boot Ftrace Patches (cont'd)

Sub-arch not quite so easy, e.g. imx:

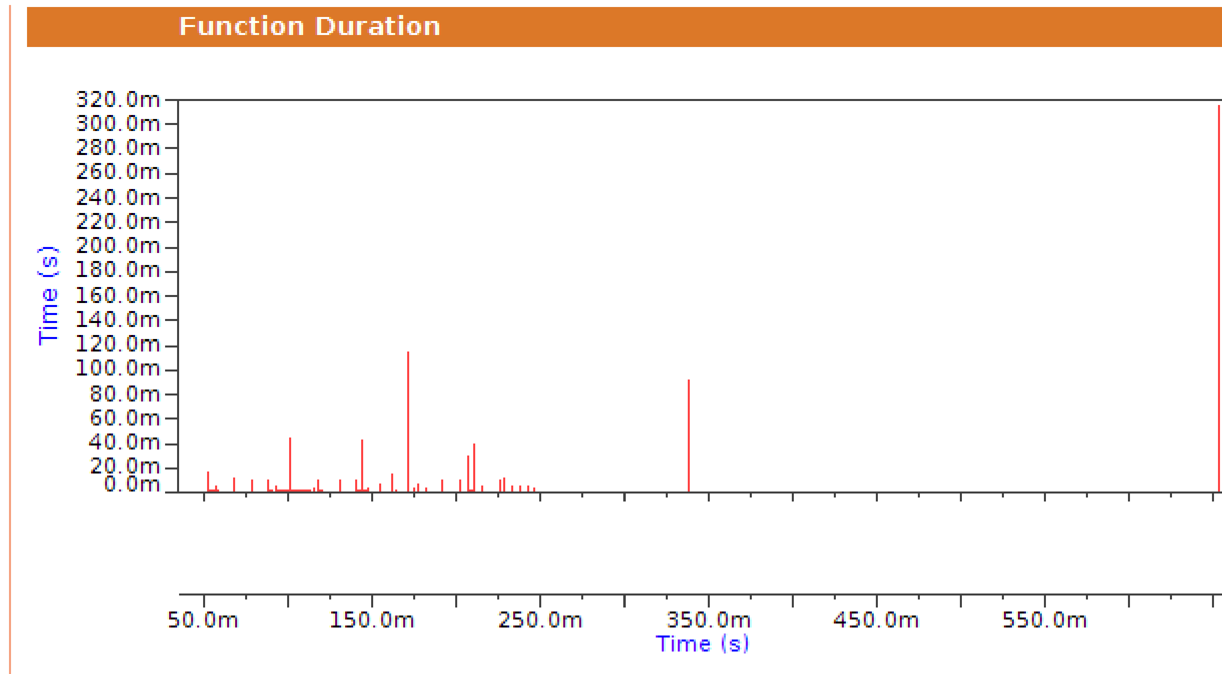
```
diff --git a/arch/arm/imx-common/timer.c b/arch/arm/imx-common/timer.c
@@ -40,7 +40,7 @@ static struct mxc_gpt *cur_gpt = (struct
-static inline int gpt_has_clk_source_osc(void)
+static inline int notrace gpt_has_clk_source_osc(void)
@@ -54,7 +54,7 @@ static inline int gpt_has_clk_source_osc
-static inline ulong gpt_get_clk(void)
+static inline ulong notrace gpt_get_clk(void)
@@ -65,7 +65,7 @@ static inline ulong gpt_get_clk(void)
 -static inline unsigned long long tick_to_time(unsigned long long tick)
+static inline unsigned long long notrace tick_to_time(unsigned long long tick)
@@ -75,7 +75,7 @@ static inline unsigned long long tick_to
-static inline unsigned long long us_to_tick(unsigned long long usec)
+static inline unsigned long long notrace us_to_tick(unsigned long long usec)
@@ -85,7 +85,7 @@ static inline unsigned long long us_to_t
-int timer_init(void)
+int notrace timer_init(void)
@@ -128,7 +128,7 @@ int timer_init(void)
-unsigned long long get_ticks(void)
+unsigned long long notrace get_ticks(void)
@@ -158,8 +158,7 @@ ulong get_timer(ulong base)
-       unsigned long long tmp;
-       ulong tmo;
+       unsigned long long tmo, tmp;
@@ -172,7 +171,7 @@ void __udelay(unsigned long usec)
-ulong get_tbclk(void)
+ulong notrace get_tbclk(void)
```

u-boot Ftrace Patches (cont'd)

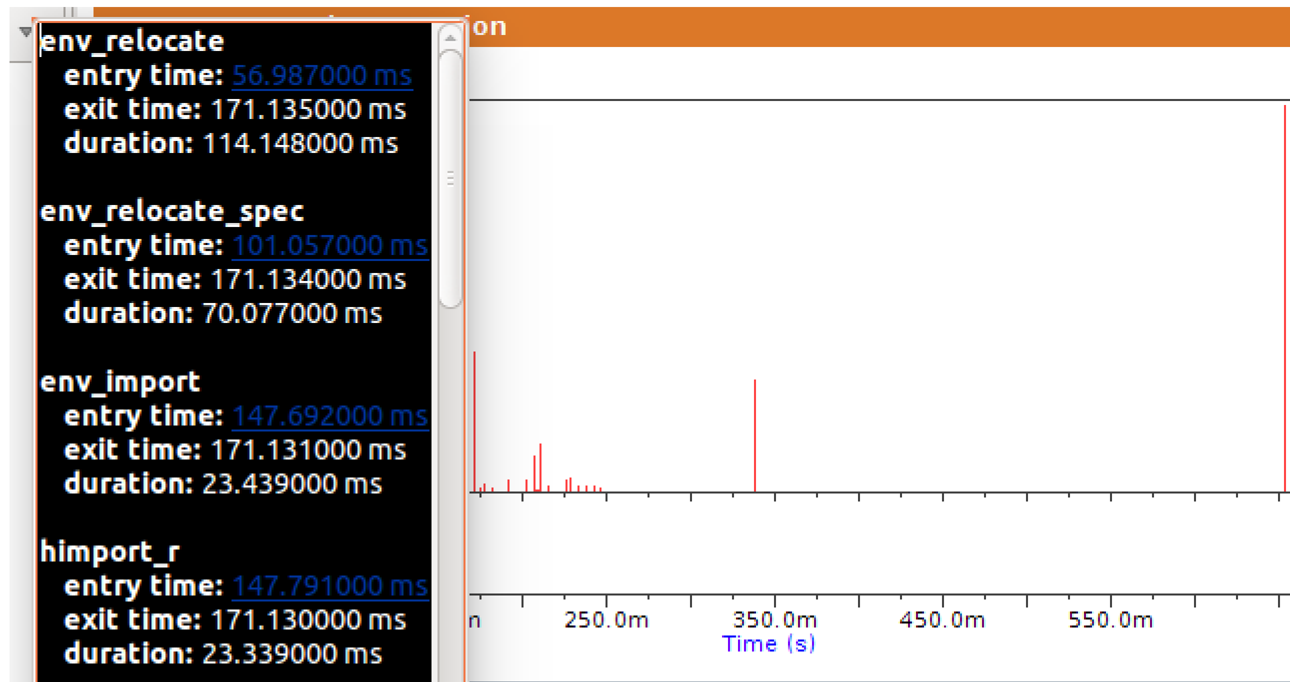
And the stinker!

```
diff --git a/lib/div64.c b/lib/div64.c
@@ -17,8 +17,10 @@
  */

 #include <linux/types.h>
+#include <compiler.h>

-uint32_t __div64_32(uint64_t *n, uint32_t base)
+/* called by tick_to_time() via do_div() when dividend is
large enough, thus notrace */
+notrace uint32_t __div64_32(uint64_t *n, uint32_t base)
 {
        uint64_t rem = *n;
        uint64_t b = base;
```

U-boot Long Poles



U-boot Long Poles



Profiling Kernel

- CONFIG_PRINTK_TIME
  - Or use “printk.time” on kernel command line
- Grabserial
- Ftrace
- LTTng
- …

CONFIG_PRINTK_TIME

```
[    . . .      ]
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] L310 cache controller enabled
[    0.000000] l2x0: 16 ways, CACHE_ID 0x410000c7, AUX_CTRL 0x32070000, Cache size: 
1048576 B
[    0.000000] sched_clock: 32 bits at 3000kHz, resolution 333ns, wraps every 1431655ms
[    0.000791] CPU identified as i.MX6Q, silicon rev 1.2
[    0.000888] Console: colour dummy device 80x30
[    0.000910] Calibrating delay loop... 1581.05 BogoMIPS (lpj=7905280)
[    0.090159] pid_max: default: 32768 minimum: 301
[    0.090290] Security Framework initialized
[    0.090304] SELinux:  Initializing.
[    0.090404] Mount-cache hash table entries: 512
[    0.091133] Initializing cgroup subsys memory
[    0.091164] Initializing cgroup subsys devices
[    0.091175] Initializing cgroup subsys freezer
[    0.091184] Initializing cgroup subsys blkio
[    0.091193] Initializing cgroup subsys perf_event
[    . . .      ]
```

Grabserial

```
[5.495787 0.002781] [    3.670201] VFS: Mounted root (ext2 filesystem) on device 179:2.
[5.499665 0.003878] [    3.679984] devtmpfs: mounted
[5.501625 0.001960] [    3.683374] Freeing unused kernel memory: 472K (c0b18000 - c0b8e000)
[5.760650 0.259025]
[5.761368 0.000718] Welcome to Mentor Embedded Linux 2014.12+snapshot-20141222 (dizzy)!
[5.767265 0.005897]
[6.309478 0.542213] [  OK  ] Set up automount Arbitrary Executab...ats File System Automount Point.
[6.333415 0.023937] [  OK  ] Reached target Swap.
[6.409136 0.075721] [  OK  ] Created slice Root Slice.
[6.442217 0.033081] [  OK  ] Listening on Journal Socket (/dev/log).
[6.461864 0.019647] [  OK  ] Listening on udev Control Socket.
[6.481365 0.019501] [  OK  ] Listening on udev Kernel Socket.
[6.501109 0.019744] [  OK  ] Listening on Journal Socket.
[6.524209 0.023100] [  OK  ] Created slice System Slice.
```

Kernel Ftrace

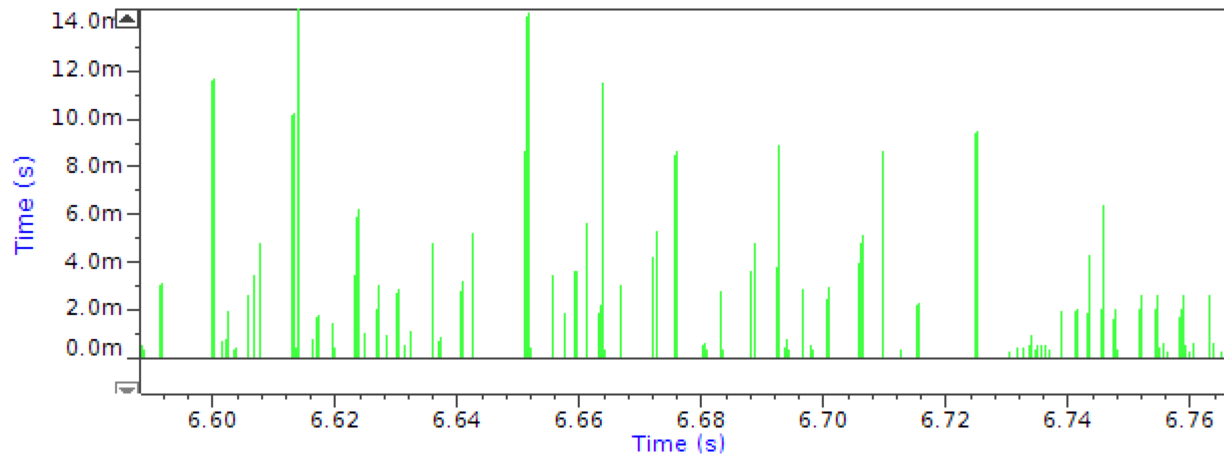
- Function_graph_tracer keeps track of time in functions
- Enabling (config frag)

  CONFIG_FTRACE=y

  CONFIG_FUNCTION_TRACER=y

  CONFIG_FUNCTION_GRAPH_TRACER=y
- Also controlled via /sys/kernel/debug/tracing
- See [kernel-source]/Documentation/trace/ftrace.txt for more information
- Only valid during/after initcalls
- See KFT if you need earlier tracing

Kernel Ftrace



Other Profiling tools

- Bootchart
  - Statistical profiling
  - Integrated with systemd
  - Helps show if there is opportunity for parallel init
  - Mostly useful for userland initialization
- Perf, operf
  - Uses H/W counters that are available in most modern SOCs
  - Low overhead – no daemons or kernel modules
  - Easy to get started, but thin on documentation
- Oprofile
  - Statistical profiler which grabs info about currently executing program when specified events occur
  - Useful for finding "hotspots"
  - Can profile both kernel and application code

Bootchart

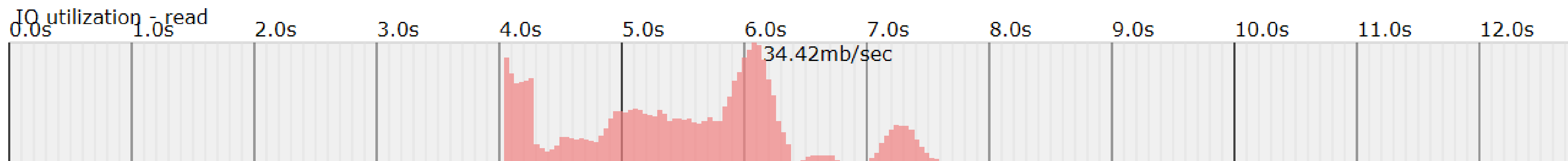
Bootchart for mx6q - Wed, 11 Feb 2015 15:09:25 +0000

System: Linux 3.10.61-mel-fsl-imx6 #2 SMP PREEMPT Thu Jan 15 09:11:56 MST 2015 armv7l
CPU: ARMv7 Processor rev 10 (v7l)
Disk: Unknown
Boot options: console=ttymxc1,115200 root=/dev/mmcblk3p1 rootwait rw video=mxcfb0:dev=ldb,LDB-XGA,if=RGB666 video=mxcfb1:dev=ldb,LDB-XGA,i
Build: Mentor Embedded Linux 2014.12+snapshot-20150112 (dizzy)
Log start time: 4.043s
Idle time: 7.666s
Graph data: 25.000 samples/sec, recorded 500 total, dropped 0 samples, 291 processes, 133 filtered

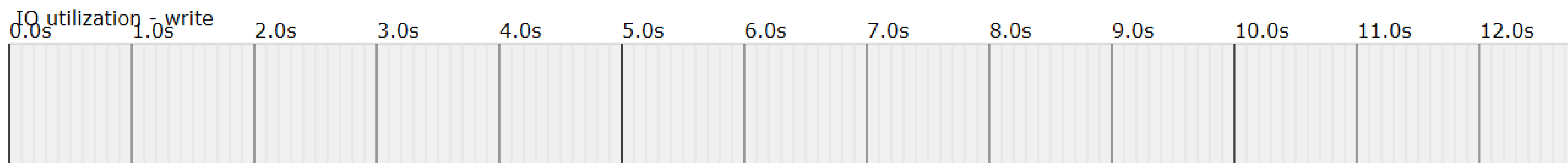
Top CPU consumers:

3.692s - systemd [1]
2.817s - systemd-bootcha [152]
0.975s - systemd-journal [177]
0.470s - mmcqd/3 [98]
0.430s - systemd-udevd [191]
0.338s - pulseaudio [244]
0.219s - systemd-udevd [216]
0.177s - systemd-udevd [209]
0.126s - dbus-daemon [222]
0.104s - connmand [254]

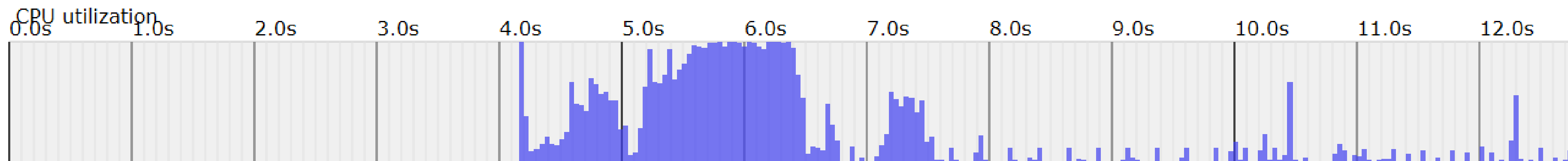
IO utilization - read



IO utilization - write



CPU utilization



CPU wait

Optimization

- Specific H/W factors will determine some methods you can or can’t use
- Focus on the fast path
- Optimize the “long poles”
- In general, I/O is the limiting factor, not cpus/cycles
  - Spend time minimizing and combining I/O
- Each product has unique attributes, and will have unique opportunities for improvement
- Some optimization techniques are broadly applicable

Optimizing U-Boot

- U-Boot must be relocated from Flash into DRAM
  - Reducing the image size reduces relocation time
- Lots of useful development functionality
  - tftp, pci scan, mem utils, disk utils, load*, dhcp, etc
  - In a production system, many of these features are unnecessary
  - Disabling these features can have a significant impact on boot time
- SPL/Falconboot may be the best for production
- Use Ftrace to find long running functions that may not be necessary to your production system
- Decompress kernel in parallel with loading
- Put lpj in kernel cmdline

Optimizing the Linux Kernel

- Size matters
  - The kernel needs to be loaded from FLASH into RAM
  - Smaller == faster
- Consider using an uncompressed kernel
  - Depends on relative speeds of flash DMA and CPU processing
- Configure as many drivers as possible as modules
  - Success depends on how well the semi vendor did their job
  - Concentrate on the largest modules
- Consider using deferred initcalls patch
  - Defer module init until much later in boot cycle
  - Initcalls deferred until triggered in userspace
- Share initializations with bootloader when possible

Optimizing the Kernel (cont'd)

- Utilize the bootloader inits when possible
  - UBI attach
  - Shared logging
  - Kernel memory map
- XIP (Your mileage may vary)
- Limit serial log prints
- Pre-configure or eliminate udev
- RTC_nosync
- Checkpoint restart
- Use parallelism for multi-core
- Cache systemd config
- Memory size optimization

Using initcall_debug

- Driver initialization calls (initcalls) spend considerable time on kernel bootup
- A kernel flag to enables initcall information during startup
- Activating: On the command line, add “initcall_debug=1”
- NOTES:
  - Increase the printk log buffer size in kernel config:
    - LOG_BUF_SHIFT=18 (256KB)
  - Turn on CONFIG_KALLSYMS (to see function names)
- After booting info is found in bootlog (dmesg)
- May identify:
  - Opportunities for parallelism
  - Opportunities for removing functionality
  - Opportunities for deferring init

Userspace – Optimize init

- Use BusyBox – Very popular in embedded systems
- Consider a custom init for very aggressive boot times
  - Can configure init=myinit on kernel command line
  - Allows complete customization of userland initialization
  - It is always faster to run native code than scripts
    - In general, every line of a script causes fork()/exec()
  - Often used in fixed function types of devices
- If you're using ready-made startup scripts
  - Eliminate unnecessary stuff (set -x)
  - Run multiple scripts in parallel wherever possible
    - May require adding some synchronization between services you start in parallel if there are dependencies.

SysV init vs systemd

- SysV is very simple and straightforward
  - Almost always faster for small embedded system
- systemd more overhead, but improves parallelism in starting many services
  - Packed with functionality, intended for large multiuser systems
  - Dependency tree orders the init scripts (unit files).
  - Win for larger systems like IVI

Optimizing systemd

- Systemd spends a lot of time scanning files and creating dependency tree
  - Likely this does not change for most embedded devices, so save the data and reload from that.
- Minimize cgroups, join cgroup controllers
  - Minimizes time to mount cgroup vfs
  - Can also enumerate cgroups with a file
  - Mount cgroups with a thread
- Disable generators
- Systemd-readahead

Using strace to profile applications

- Strace can be used to collect timing information for a process
  - strace –tt 2>/tmp/strace.log thttpd …
- Determine where time is being spent in application startup
- Can also collect system call counts (-c)
- Can see time spent in each system call (-T)
- Great for finding extraneous operations (ubiquitous)
  - scanning invalid paths for files (e.g.dynamic libs, fonts, etc),
  - opening a file multiple times, etc.
- Strace can follow children (-o -ff)
- Strace adds SIGNIFICANT overhead to the execution of the program
  - Good for <u>relative</u> timings, not <u>absolute</u>
  - May slow execution so much that it “breaks” interaction with other processes

Using ftrace for profiling

- Mainlined in 2.6.27
- Derived from RT-preempt latency tracer
- Instrumentation
  - Explicit
    - Tracepoints defined by declaration (manual instrumentation)
    - Calls to trace handler written in source code
  - Implicit
    - Automatically inserted by compiler
      - Uses gcc '-pg' option
- Controlled via /sys/kernel/debug/tracing
- See [kernel-source]/Documentation/trace/ftrace.txt for more information

Using systemd

- Alternative to SysV Init
- Avoids much fork/exec of typical start up scripts
- Compatible with SysV Init scripts, but
  - Translate to systemd config files for best results
- Solves many of the startup dependencies quite nicely
- Still somewhat experimental – works well, but might not be integrated with everything you need
- See http://0pointer.de/blog/projects/systemd.html for an interesting introduction by the author

Application Prelink

- A good portion of application initialization time is spent resolving symbols to dynamic libraries
- Using Prelinking you can cut off a significant portion of application startup time if you have a large/complicated userland
- Tries to assign a preferred address space to each library used by an application – ahead of time
- Prelink can be enabled during construction of rootfs by Yocto-based distributions

Design your Apps for Fastboot

- Keep it small
- Prelink
- Be careful adding dependencies on new libraries
  - it can snowball
- Keep fork/exec to minimum
- Some multithreading can help esp. with multicore
- Use Analysis tools – profiling, strace, MESA, etc.
- Use only Fast-path I/O Peripherals (e.g. – no Wifi)
- Avoid "discovery" code if possible

© 2012 MontaVista Software - Confidential

Much More ! (background slide)

- XIP
- Limit console printk()
- Preconfigure udev
- Kernel modules
- Multipage DMA (scatter-gather)
- RTC_nosync
- Checkpoint restart
- Use parallelism for multi-core
- Cache systemd config
- ...