init 빨리 실행하기

#ARM_Target_Board
#deferred_initcall_patch

구본규
protocolstack9@gmail.com

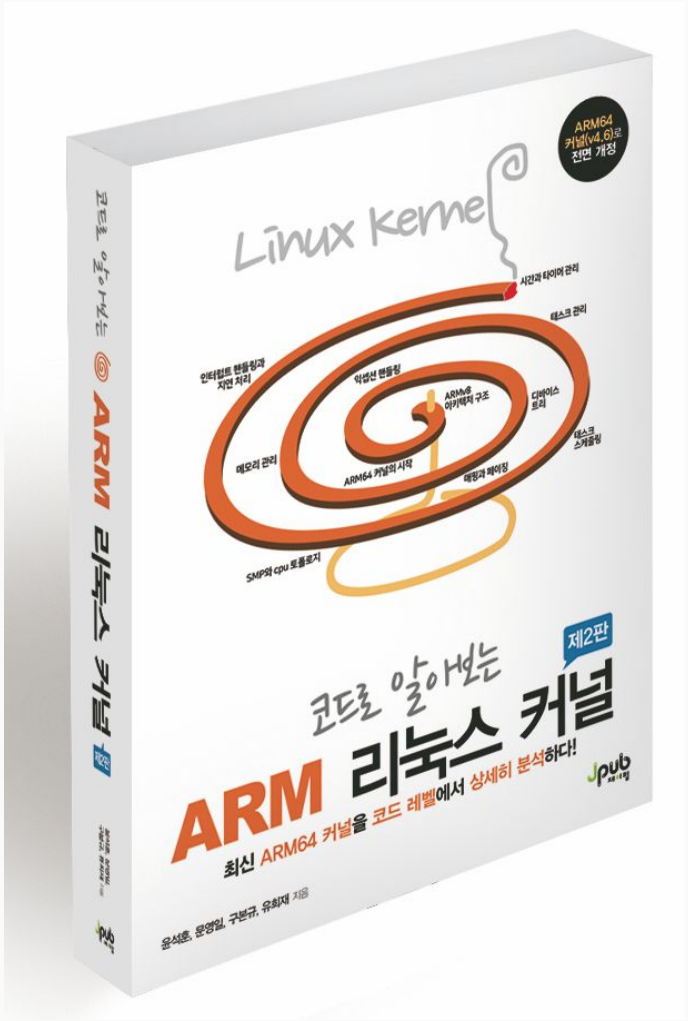
I am ...

- 임베디드 엔지니어

- iamroot.org ARM 스터디 9차

- 코드로 알아보는 ARM 리눅스 커널 2판

- 관심분야
  - architecture inside
  - deep learning & robot



작업 목적

차량용 솔루션에 사용 (ex. 후방 카메라)

작업 목적

차량용 솔루션에 사용 (ex. 후방 카메라)

최대한 빨리 부팅해서 화면을 그리는 프로그램을 실행하자

quickboot 솔루션

boot as quickly as possible!

다양한 임베디드 디바이스에서 사용 (스마트폰, 셋톱박스, 카메라, 의료기기, 오토모티브, ...)

quickboot 솔루션

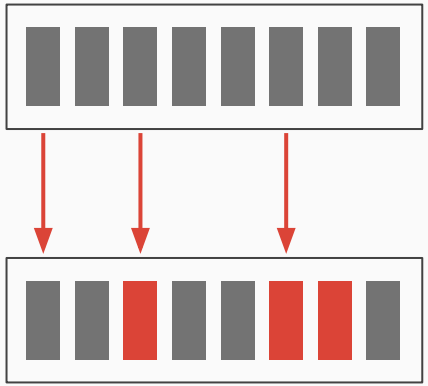
플랫폼에 따라 다양한 솔루션 선택 가능

- 범용성 vs 최적화
- 구현시간 vs 부팅단축시간
- 업데이트 용이성, 하드웨어 구성 등

몇 가지 기법을 섞어 사용

quickboot 솔루션 예시

- hibernation & loading



- hibernation: suspend to disk
- 페이지를 필요 시점에 따라 구분
  preloading pages : 필수 페이지
  normal pages : 그외 페이지
  (배치 로딩 / 익셉션 로딩)
- 스토리지 성능 최적화 + burst 로딩

quickboot 솔루션 예시

- SMP 전담 부팅

QuickBoot firmware 담당
- 특정 디바이스 초기화 (ex. 카메라, 디스플레이)
- QuickBoot Linux 로딩

지향점

기본 BSP 구조는 유지 (별도 SMP 펌웨어 x, 하이버네이션 x)

지향점

기본 BSP 구조는 유지 (별도 SMP 펌웨어 x, 하이버네이션 x)

포팅과 디버깅은 쉽도록

환경

kernel v4.14.73

u-boot 2018-05

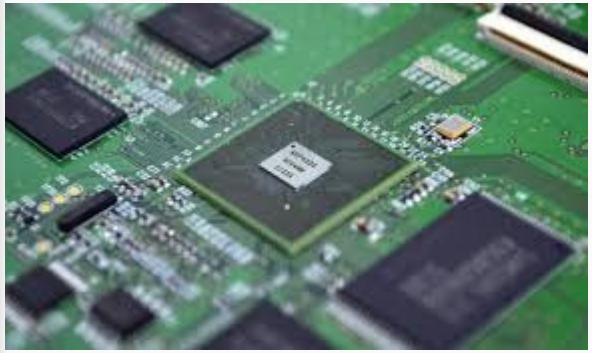
CPU : nxp3220 (cortex-a7x2 800MHz)

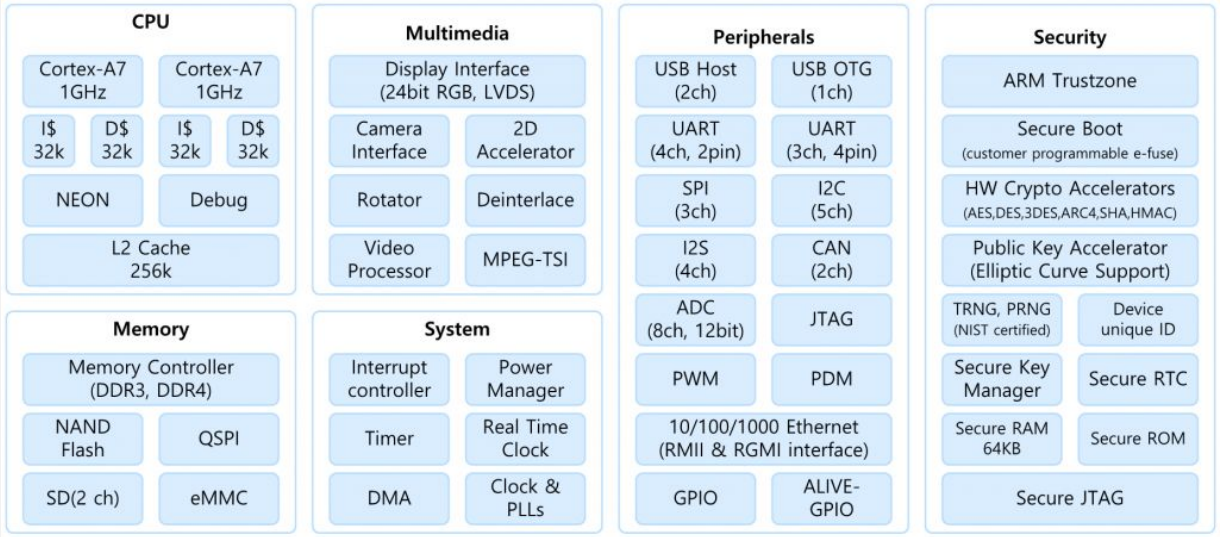
Memory : ddr3

Storage : eMMC, SD

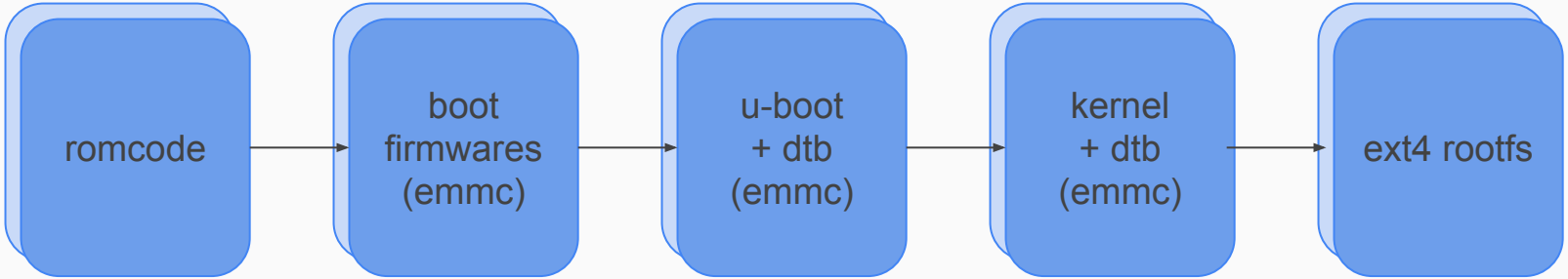
no dvfs

no suspend/resume





타겟보드 부팅과정


romcode
boot
firmwares
(emmc)
u-boot
+ dtb
(emmc)
kernel
+ dtb
(emmc)
ext4 rootfs

작업과정

1) 프로파일링
2) u-boot, kernel 다이어트
3) base: deferred initcalls
4) more deferred initcalls
5) initcall 함수 지연 + a
6) 커널 이미지 타입 선택
7) more and more deferred initcalls
8) kernel CONFIG_HZ 변경

1) 프로파일링

메시지 출력 시간 및 간격 측정 (작업과정마다 측정)

ex. grabserial

```
$ sudo grabserial -d /dev/ttyUSB0 -t -m "Starting"
[2.801862 0.000058] Starting kernel …
[2.802296 0.000434]
[2.961674 0.159378] Uncompressing Linux... done, booting the kernel.
[6.240257 3.278583] Starting logging: OK
[6.272364 0.032107] Starting mdev...
```

출력 시간 출력 간격

1) 프로파일링

커널 bootargs) printk.time=1

```
[    0.000091] Switching to timer-based delay loop, resolution 10ns
[    0.000381] Console: colour dummy device 80x30
[    0.000413] Calibrating delay loop (skipped), value calculated using timer
frequency.. 200.00 BogoMIPS (lpj=100000)
[    0.000429] pid_max: default: 32768 minimum: 301
[    0.000563] Security Framework initialized
[    0.000703] AppArmor: AppArmor disabled by boot time parameter
```

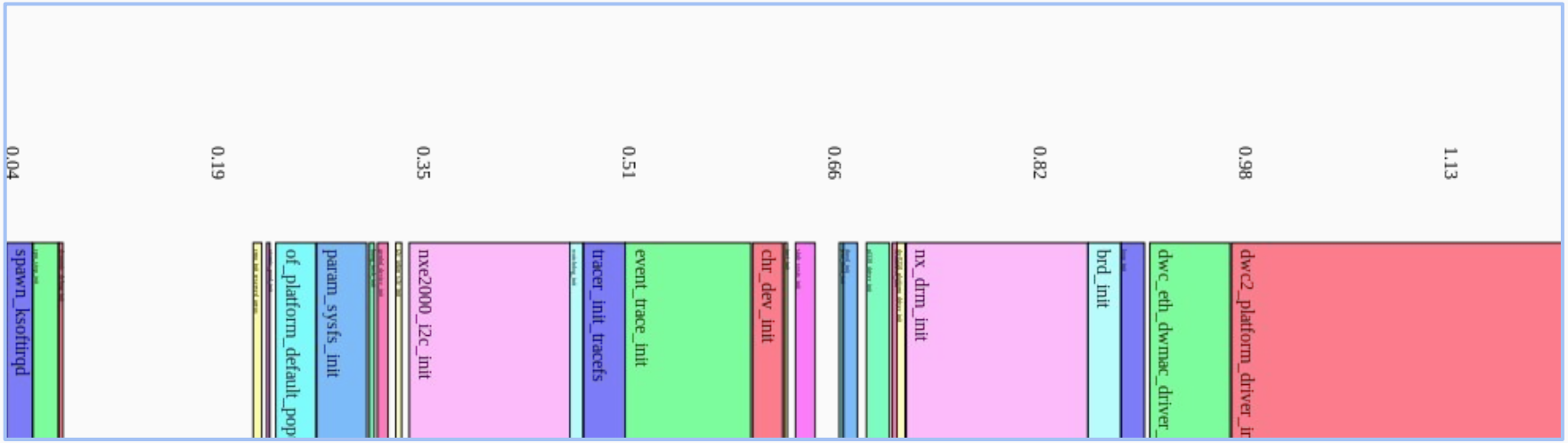
1) 프로파일링

커널 bootargs) initcall_debug=1

```
[    0.040237] calling  cpu_suspend_alloc_sp+0x0/0x7c @ 1
[    0.040258] initcall cpu_suspend_alloc_sp+0x0/0x7c returned 0 after 0 usecs
[    0.040275] calling  init_static_idmap+0x0/0xf8 @ 1
[    0.040312] Setting up static identity map for 0x48100000 - 0x48100060
[    0.040350] initcall init_static_idmap+0x0/0xf8 returned 0 after 0 usecs
[    0.040367] calling  spawn_ksoftirqd+0x0/0x54 @ 1
[    0.060151] initcall spawn_ksoftirqd+0x0/0x54 returned 0 after 19531 usecs
```

1) 프로파일링

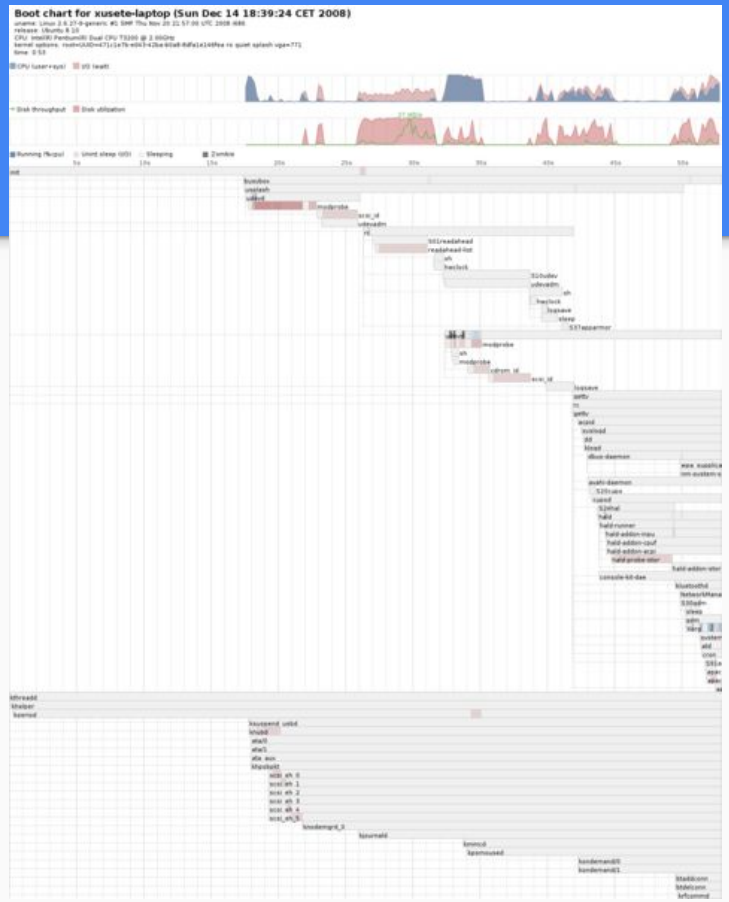
dmesg | perl scripts/bootgraph.pl > output.svg



1) 프로파일링

bootchart - 유저스페이스 부트 시퀀스

systemd-analyze - systemd 사용 시



2) u-boot, kernel 다이어트

[공통]

사용하지 않는 설정 제거 (미사용 장치 드라이버, 프로파일링/디버깅, ...)

[u-boot]

메시지 출력 제거, CONFIG_SYS_DCACHE_OFF 제거, BOOTDELAY=0

[kernel]

메시지 출력 제거 (bootargs: quiet)

3) base: deferred initcalls

- initcall mechanism

  커널 초기화 과정에 함수를 추가하고, 호출 순서를 부여

- __init/__initdata 매크로

  함수가 배치되는 섹션을 지정하고 부트 마지막에 free

3) base: deferred initcalls

패치 적용 https://elinux.org/Deferred_Initcalls

- First submitted 9/2008 by Tim Bird
- Updated 7/2013 by Alexandre Belloni

3) base: deferred initcalls

[patch concept] initcall 함수를 지연시키자

3) base: deferred initcalls

[patch concept] initcall 함수를 지연시키자

오래 걸리지만 미룰 수 있는 initcall 함수 (= 미룰 가치가 있는 함수)

3) base: deferred initcalls

언제까지?

3) base: deferred initcalls

언제까지?

init 프로세스 실행 이후까지 (= userspace가 실행될 때까지)

initcall 관련 파일들 (패치 파일)

- initcall 실행
  <init/main.c>
- 관련 매크로
  <include/linux/init.h>
  <include/linux/module.h>
- init 섹션 배치 script (결과 System.map)
  <arch/arm/kernel/vmlinux.lds.S>
  <include/asm-generic/vmlinux.lds.h>
- 지연 실행 함수 호출부
  <fs/proc/root.c>

initcall levels 추가

init process 실행 이후 지연시킨 initcall 함수들을 실행

```
<default levels>

early(0)
core(1)
postcore(2)
arch(3)
subsys(4)
fs(5)
rootfs(rootfs)
device(6)
late(7)
init process
```

```
<deferred levels>

early(0)
core(1)
postcore(2)
arch(3)
subsys(4)
fs(5)
rootfs(rootfs)
device(6)
late(7)
init process
deferred_initcalls
```

initcall 섹션 추가

```
#define INIT_CALLS
      VMLINUX_SYMBOL(__initcall_start) = .;
      INIT_CALLS_LEVEL(0)
      INIT_CALLS_LEVEL(1)
      INIT_CALLS_LEVEL(2)
      INIT_CALLS_LEVEL(3)
      INIT_CALLS_LEVEL(4)
      INIT_CALLS_LEVEL(5)
      INIT_CALLS_LEVEL(rootfs)
      INIT_CALLS_LEVEL(6)
      INIT_CALLS_LEVEL(7)
      VMLINUX_SYMBOL(__initcall_end) = .;
```

```
#define INIT_CALLS
      VMLINUX_SYMBOL(__initcall_start) = .;
      INIT_CALLS_LEVEL(0)
      INIT_CALLS_LEVEL(1)
      INIT_CALLS_LEVEL(2)
      INIT_CALLS_LEVEL(3)
      INIT_CALLS_LEVEL(4)
      INIT_CALLS_LEVEL(5)
      INIT_CALLS_LEVEL(rootfs)
      INIT_CALLS_LEVEL(6)
      INIT_CALLS_LEVEL(7)
      VMLINUX_SYMBOL(__initcall_end) = .;
      DEFERRED_INITCALLS
```

```
#define INIT_CALLS_LEVEL(level)
    VMLINUX_SYMBOL(__initcall##level##_start)=.;
    KEEP(*(.initcall##level##.init))
    KEEP(*(.initcall##level##s.init))
```

```
#define DEFERRED_INITCALLS
    VMLINUX_SYMBOL(__deferred_initcall_start)=.;
    *(.deferred_initcall.init)
    VMLINUX_SYMBOL(__deferred_initcall_end)=.;
```

init 실행 과정

```
<init/main.c>

start_kernel()
     …
     rest_init();
          kernel_thread(kernel_init…)
          …
```

```
kernel_init()
     kernel_init_freeable();
     free_initmem();
     mark_readonly();
     …
     run_init_process();
```

```
kernel_init_freeable()
     …
     do_basic_setup()
          …
          do_initcalls();
               for 0~initcall_levels-1
                    do_initcall_level();
     …
```

init 실행 수정 (deferred initcall)

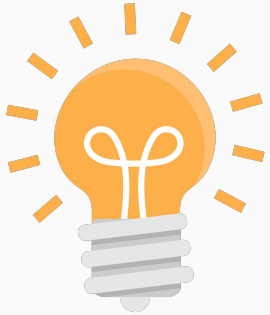
```
<init/main.c>

start_kernel()
     …
     rest_init();
          kernel_thread(kernel_init…)
          …
```

```
kernel_init()
     kernel_init_freeable();
     /*free_initmem();*/
     mark_readonly();
     …
     run_init_process();
```

```
kernel_init_freeable()
     …
     do_basic_setup()
          …
          do_initcalls();
               for 0~initcall_levels-1
                    do_initcall_level();
     …
```

```
/* call deferred initcalls */
do_deferred_initcalls()    // by read_proc
     for __deferred_initcall_start
          ~ __deferred_initcall_end
     free_initmem()
```

4) more deferred initcalls



- fs(5)와 device(6) 사이에서 실행 할 함수 (ex. emmc init)
- 지연 실행에도 레벨을 두자
- late_initcall은 항상 마지막에 실행

  (-EPROBE_DEFER를 처리하는 deferred_probe_initcall)

initcall levels 변경 (more deferred initcalls)

```
<deferred levels>

early(0)
core(1)
postcore(2)
arch(3)
subsys(4)
fs(5)
rootfs(rootfs)
device(6)
late(7)
init process
deferred_initcalls
```

```
<more deferred levels>

early(0)
core(1)
postcore(2)
arch(3)
subsys(4)
fs(5)
rootfs(rootfs)
early_device(6)
device(7)
init process
deferred_initcall0
deferred_initcall1
deferred_initcall2
late(8)
```

initcall 섹션 추가 (more deferred initcalls)

```
<deferred levels>

#define INIT_CALLS
    VMLINUX_SYMBOL(__initcall_start) = .;
    INIT_CALLS_LEVEL(0)
    INIT_CALLS_LEVEL(1)
    INIT_CALLS_LEVEL(2)
    INIT_CALLS_LEVEL(3)
    INIT_CALLS_LEVEL(4)
    INIT_CALLS_LEVEL(5)
    INIT_CALLS_LEVEL(rootfs)
    INIT_CALLS_LEVEL(6)
    INIT_CALLS_LEVEL(7)
    VMLINUX_SYMBOL(__initcall_end) = .;
    DEFERRED_INITCALLS
```

```
<more deferred levels>

#define INIT_CALLS
    VMLINUX_SYMBOL(__initcall_start) = .;
    INIT_CALLS_LEVEL(0)
    INIT_CALLS_LEVEL(1)
    INIT_CALLS_LEVEL(2)
    INIT_CALLS_LEVEL(3)
    INIT_CALLS_LEVEL(4)
    INIT_CALLS_LEVEL(5)
    INIT_CALLS_LEVEL(rootfs)
    INIT_CALLS_LEVEL(6)
    INIT_CALLS_LEVEL(7)
    INIT_CALLS_LEVEL(8)
    VMLINUX_SYMBOL(__initcall_end) = .;
    DEFERRED_INIT_CALLS_LEVEL(0)
    DEFERRED_INIT_CALLS_LEVEL(1)
    DEFERRED_INIT_CALLS_LEVEL(2)
    VMLINUX_SYMBOL(__deferred_initcall_end) = .;
```

init 실행 수정 (more deferred initcalls)

```
<init/main.c>

start_kernel()
     …
     rest_init();
          kernel_thread(kernel_init…)
          …
```

```
kernel_init()
     kernel_init_freeable();
     /*free_initmem();*/
     /*mark_readonly();*/
     …
     run_init_process();
```

```
kernel_init_freeable()
     …
     do_basic_setup()
          …
          do_initcalls();
               for 0 ~ initcall_levels
                       -DEFERED_LEVEL-1
                    do_initcall_level();
     …
```

```
/* call deferred initcalls */
do_deferred_initcalls()    // by thread
     for initcall_levels-DEFERRED_LEVEL
          ~ __deferred_initcall_end
     free_initmem()
```

5) initcall 함수 지연 + a

initcall 함수 중에서 오래 걸리는 함수들을 지연시키자

bootargs) initcall_debug=1

https://elinux.org/Initcall_Debug

5) initcall 함수 지연 + a

```
$ dmesg -s 128000 | grep "initcall" | sed "s/\(.*\)after\(.*\)/\2 \1/g" | sort -n
19531 usecs [  0.060158] initcall spawn_ksoftirqd+0x0/0x54 returned 0
19531 usecs [  0.080191] initcall cpu_stop_init+0x0/0xbc returned 0
19531 usecs [  0.268640] initcall of_platform_default_populate_init+0x0/0x80 returned 0
23156 usecs [  0.428997] initcall pl330_driver_init+0x0/0xc returned 0
26075 usecs [  0.375975] initcall dw_mmc_nexell_init+0x0/0x10 returned 0
29296 usecs [  0.298030] initcall param_sysfs_init+0x0/0x1dc returned 0
```

<수행 시간> <시작 시간> <initcall 함수>

전체를 지연시킬 수 없는 함수는 쪼개서 지연

6) 커널 이미지 타입 선택

각각 빌드해 비교

Image = load time

zImage = load time + decompress time

compression type : GZIP / LZO / LZMA / XZ / LZ4

6) 커널 이미지 타입 선택

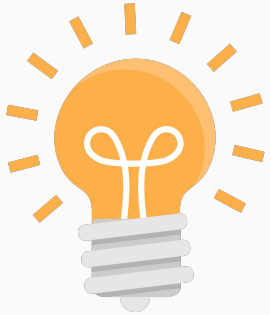
```
load time
u-boot: <fs/fs.c>
do_load() 함수 출력
    13891164 bytes read in 335 ms (39.5 MiB/s)

decompress time
kernel: <arch/arm/boot/compressed/misc.c>
decompress_kernel() // 출력문 주석처리한 시간과 비교
    putstr("Uncompressing Linux...");
    do_decompress(...);
```

6) 커널 이미지 타입 선택

결과: Image 승!!

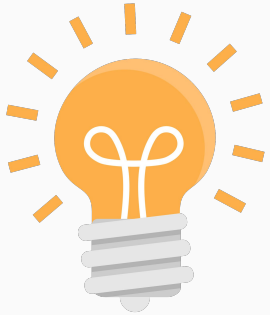
- emmc 8bit DDR은 제법 빠르다
- 커널이 다이어트를 해서 크기가 작다
- CONFIG_CC_OPTIMIZE_FOR_SIZE 를 한다면?

7) more and more deferred initcalls



init process를 조금 더 빨리 실행해보자

7) more and more deferred initcalls



init process를 조금 더 빨리 실행해보자

device 전체를 지연 실행한다면?!

initcall levels 변경 (more and more deferred initcalls)

```
<more deferred levels>

early(0)
core(1)
postcore(2)
arch(3)
subsys(4)
fs(5)
rootfs(rootfs)
early_device(6)
device(7)
init process
deferred_initcall0
deferred_initcall1
deferred_initcall2
late(8)
```

```
<more and more deferred levels>

early(0)
core(1)
postcore(2)
arch(3)
subsys(4)
fs(5)
rootfs(rootfs)
early_device(6)
init process
deferred_initcall0
deferred_initcall1
deferred_initcall2
device(7)
late(8)
```

initcall 섹션 동일 (more and more deferred initcalls)

```
<more and more deferred levels>

#define INIT_CALLS
    VMLINUX_SYMBOL(__initcall_start) = .;
    INIT_CALLS_LEVEL(0)
    INIT_CALLS_LEVEL(1)
    INIT_CALLS_LEVEL(2)
    INIT_CALLS_LEVEL(3)
    INIT_CALLS_LEVEL(4)
    INIT_CALLS_LEVEL(5)
    INIT_CALLS_LEVEL(rootfs)
    INIT_CALLS_LEVEL(6)
    INIT_CALLS_LEVEL(7)
    INIT_CALLS_LEVEL(8)
    VMLINUX_SYMBOL(__initcall_end) = .;
    DEFERRED_INIT_CALLS_LEVEL(0)
    DEFERRED_INIT_CALLS_LEVEL(1)
    DEFERRED_INIT_CALLS_LEVEL(2)
    VMLINUX_SYMBOL(__deferred_initcall_end) = .;
```

8) kernel CONFIG_HZ 변경

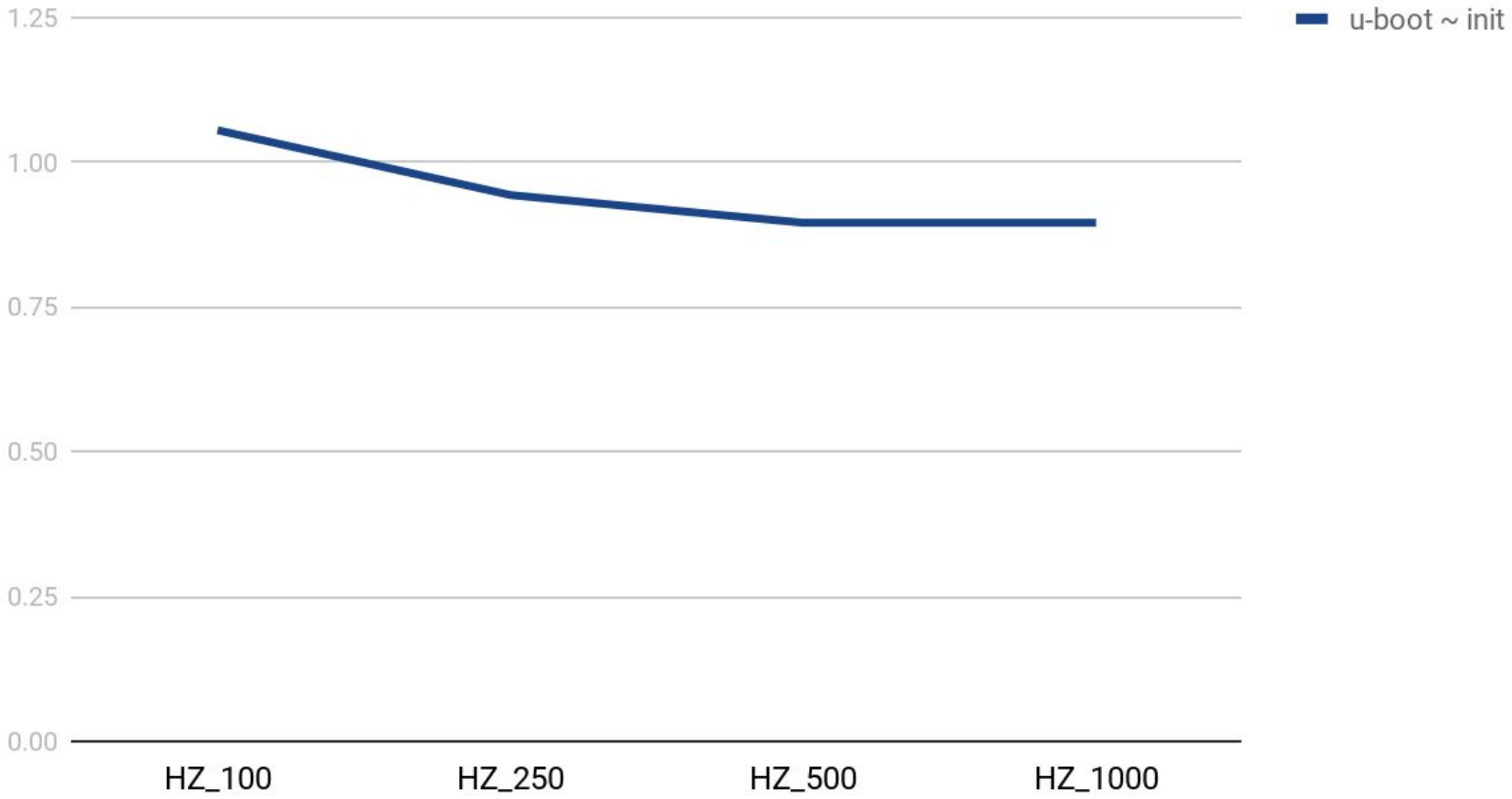
HZ_100 -> HZ_1000 변경

8) kernel CONFIG_HZ 변경

HZ_100 -> HZ_1000 변경

init 프로세스 실행시점이 빨라졌다. but why?

HZ에 따른 실행시간



Result

QuickBoot 솔루션 중 하나로 deferred initcall을 적용

Result

QuickBoot 솔루션 중 하나로 deferred initcall을 적용

최적화는 대표적인 3D 작업. 명확한 목표(요구사항 정의) 필요!

- side effect 가능성 최소화
- ex) u-boot~init 실행까지 1.2s 달성

Result

QuickBoot 솔루션 중 하나로 deferred initcall을 적용

최적화는 대표적인 3D 작업. 명확한 목표(요구사항 정의) 필요!
- side effect 가능성 최소화
- ex) u-boot~init 실행까지 1.2s 이내

config 바꾸고, initcall 지정하고, 빌드 하고, 측정하는 번거로움
- 단순 작업이 대부분의 시간 소모. 자동화의 필요성!

Q n A