05-2 Adding to the Kernel, Kernel Initialization

Adding to the Kernel

- Makefile Targets
- Kernel Configuration
- Custom Configuration Options
- Kernel Makefiles
- Kernel Documentation

Composite Kernel Image

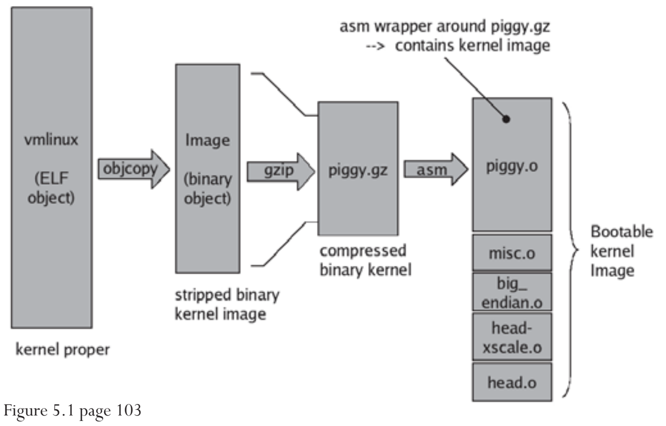


Figure 5.1 page 103

piggy.S

3.2 kernel

```
  .section .piggydata,#alloc
  .globl    input_data
input_data:
  .incbin   "arch/arm/boot/compressed/piggy.gz"
  .globl    input_data_end
input_data_end:
```

Compiling Kernel

```
host$ source ~/crossCompileEnv.sh
host$ make -j3 uImage
... < many build steps omitted for clarity >
AS      arch/arm/boot/compressed/head.o
  XZKERN  arch/arm/boot/compressed/piggy.xzkern
…
  AS      arch/arm/boot/compressed/piggy.xzkern.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.8.13+
Created:      Thu Oct  3 17:13:18 2013
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    2898464 Bytes = 2830.53 kB = 2.76 MB
Load Address: 80008000
Entry Point:  80008000
  Image arch/arm/boot/uImage is ready
```

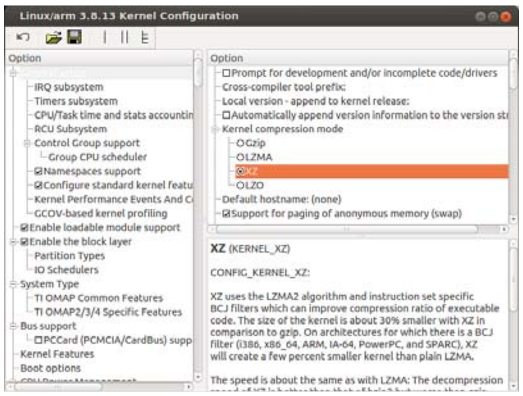
arch/arm/boot/compressed

```
host$ ls
ashldi3.o         hyp-stub.o       piggy.lzo.S
ashldi3.S         hyp-stub.S       piggy.xzkern
atags_to_fdt.c    lib1funcs.o      piggy.xzkern.o
big-endian.S      lib1funcs.S      piggy.xzkern.S
decompress.c      libfdt_env.h     sdhi-sh7372.c
decompress.o      ll_char_wr.S     sdhi-shmobile.c
head.o            Makefile         sdhi-shmobile.h
head.S            misc.c           string.c
head-sa1100.S     misc.o           string.o
head-shark.S      mmcif-sh7372.c   vmlinux
head-sharpsl.S    ofw-shark.c      vmlinux.lds
head-shmobile.S   piggy.gzip.S     vmlinux.lds.in
head-xscale.S     piggy.lzma.S
```

piggy.xzkern.S

```
  .section .piggydata,#alloc
  .globl    input_data
input_data:
  .incbin   "arch/arm/boot/compressed/piggy.xzkern"
  .globl    input_data_end
input_data_end:
```

How does it know to use kernxz?


Linux/arm 3.8.13 Kernel Configuration
Option
IRQ subsystem
Timers subsystem
CPU/Task time and stats accountin
RCU Subsystem
Control Group support
Group CPU scheduler
☑Namespaces support
☑Configure standard kernel featu
Kernel Performance Events And C
GCOV-based kernel profiling
☑Enable loadable module support
☑Enable the block layer
Partition Types
IO Schedulers
System Type
TI OMAP Common Features
TI OMAP2/3/4 Specific Features
Bus support
☐PCCard (PCMCIA/CardBus) supp
Kernel Features
Boot options
Option
☐Prompt for development and/or incomplete code/drivers
Cross-compiler tool prefix:
Local version - append to kernel release:
☐Automatically append version information to the version st
Kernel compression mode
○Gzip
○LZMA
◉XZ
○LZO
Default hostname: (none)
☑Support for paging of anonymous memory (swap)

XZ (KERNEL_XZ)

CONFIG_KERNEL_XZ:

XZ uses the LZMA2 algorithm and instruction set specific BCJ filters which can improve compression ratio of executable code. The size of the kernel is about 30% smaller with XZ in comparison to gzip. On architectures for which there is a BCJ filter (i386, x86_64, ARM, IA-64, PowerPC, and SPARC), XZ will create a few percent smaller kernel than plain LZMA.

The speed is about the same as with LZMA: The decompression

Bootstrap Loader (not bootloader)

- Provide context for kernel
  - Enable instruction set
  - Data caches
  - Disable interrupt
  - C runtime environment
- Decompress (misc.o)
- Relocate kernel image

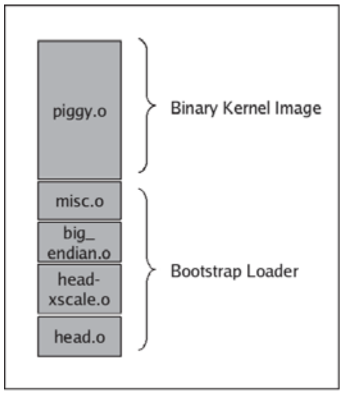


Figure 5-2 page 105

Bootstrap Loader (not bootloader)

```
 LD      vmlinux
  SORTEX  vmlinux
sort done marker at 81c420
  SYSMAP  System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  XZKERN  arch/arm/boot/compressed/piggy.xzkern
  CC      arch/arm/boot/compressed/misc.o
  CC      arch/arm/boot/compressed/decompress.o
  CC      arch/arm/boot/compressed/string.o
  SHIPPED arch/arm/boot/compressed/hyp-stub.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  AS      arch/arm/boot/compressed/ashldi3.o
  AS      arch/arm/boot/compressed/hyp-stub.o
  AS      arch/arm/boot/compressed/piggy.xzkern.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
```

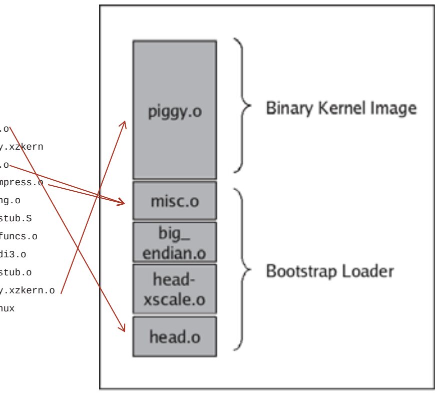
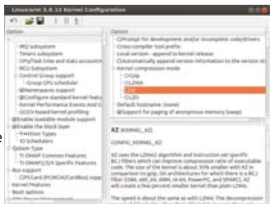


Figure 5-2 page 105

decompress.c



```
#ifdef CONFIG_KERNEL_GZIP
#include "../../../../lib/decompress_inflate.c"
#endif

#ifdef CONFIG_KERNEL_LZO
#include "../../../../lib/decompress_unlzo.c"
#endif

#ifdef CONFIG_KERNEL_LZMA
#include "../../../../lib/decompress_unlzma.c"
#endif

#ifdef CONFIG_KERNEL_XZ
#define memmove memmove
#define memcpy memcpy
#include "../../../../lib/decompress_unxz.c"
#endif
```

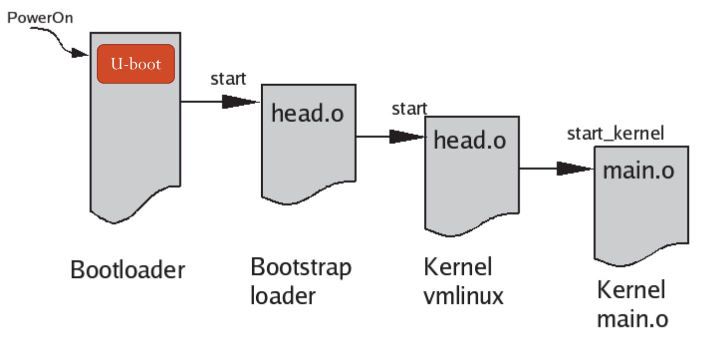
Boot Messages

- See handout
- Note *kernel version string*
- Note *kernel command line*
- *EBC Boot Sequence* shows how to display the messages in the handout

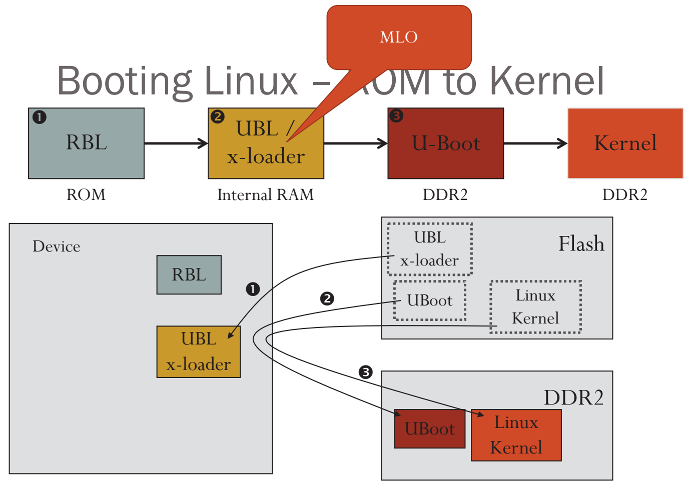
```
bone$ cd /boot
bone$ ls -F
config-3.8.13-bone64     SOC.sh                   uEnv.txt
dtbs/                    System.map-3.8.13-bone64 vmlinuz-3.8.13-
bone64*
initrd.img-3.8.13-bone64  uboot/
bone$ cat uEnv.txt
#Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0
uname_r=3.8.13-bone64
cmdline=quiet init=/lib/systemd/systemd
```

remove

5-3 ARM boot control flow



Booting Linux – ROM to Kernel



.../arch/arm/boot/compressed/head.S

How do you find the value?

```
#include <linux/linkage.h>

#ifdef DEBUG

#if defined(CONFIG_DEBUG_ICEDCC)

#ifdef CONFIG_CPU_V6
          .macro    loadsp, rb
          .endm
          .macro    writeb, ch, rb
          mcr       p14, 0, \ch, c0, c5, 0
          .endm
#else
          .macro    loadsp, rb
          .endm
          .macro    writeb, ch, rb
          mcr       p14, 0, \ch, c1, c0, 0
          .endm
#endif
```

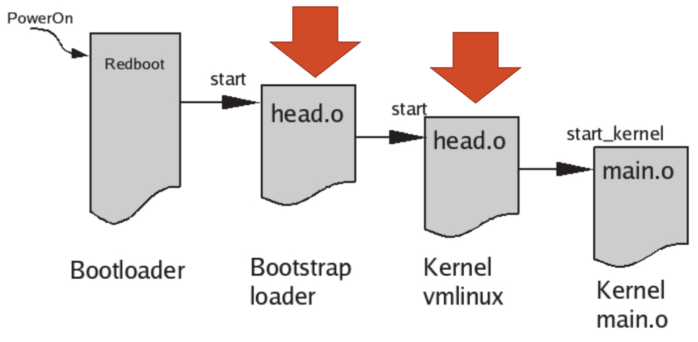
```
#
#else

#include <mach/debug-macro.S>

          .macro    writeb,   ch, rb
          senduart \ch, \rb
          .endm

#if defined(CONFIG_ARCH_SA1100)
          .macro    loadsp, rb
          mov       \rb, #0x80000000    @
    physical base address
```

2 head.o's



.../arch/arm/kernel/head.S

1. Checks for valid processor and architecture
2. Creates initial page table entries
3. Enables the processor's memory management unit (MMU)
4. Establishes limited error detection and reporting
5. Jumps to the start of the kernel proper, **`start_kernel()`** in **`main.c`**.

Find these on the handout

Kernel Startup

- **arch/arm/kernel/head.S**

`b start_kernel`

Find this for HW 6

.../init/main.c

```
asmlinkage void __init start_kernel(void)
{
   char * command_line;
   extern struct kernel_param __start___param[], __stop___param[];

   smp_setup_processor_id();

   /*
    * Need to run as early as possible, to initialize the
    * lockdep hash:
    */
   lockdep_init();
   debug_objects_early_init();
   cgroup_init_early();

   local_irq_disable();
   early_boot_irqs_off();
   early_init_irq_lock_class();
```

Kernel Command Line Processing

- Read 5.3 on Kernel Command-Line Processing
- It presents the `__setup` macro

```
console=ttyO0,115200n8
  run_hardware_tests
  root=/dev/mmcblk0p2 ro
  rootfstype=ext4 rootwait
```

Console Setup Code Snippet

```
/*
* Setup a list of consoles. Called from init/main.c
```

```
#ifdef CONFIG_SERIAL_OMAP
          if (!strncmp(str, "tty0", 4) && '0' <= str[4] && '9' >= str[4]) {
                    str[3] = 'O';
                    pr_warn("We are opening your eyes, assuming you want to
use an OMAP based serial driver and not a zeroMAP based one! ;)\n");
                    pr_warn("Which means 'tty0%s' was changed to 'ttyO%s'
automagically for your pleasure.\n", str+4, str+4);
          }
#endif
```

```
...
return 1;
}
__setup("console=", console_setup);
```

Registration function

From .../kernel/printk.c

.../include/linux/init.h

```
/*
 * Only for really core code.  See moduleparam.h for the normal way.
 *
 * Force the alignment so the compiler doesn't space elements of the
 * obs_kernel_param "array" too far apart in .init.setup.
 */
#define __setup_param(str, unique_id, fn, early)            \
   static char __setup_str_##unique_id[] __initdata __aligned(1) = str;
                                                             \
   static struct obs_kernel_param __setup_##unique_id       \
         __used __section(.init.setup)                      \
         __attribute__((aligned((sizeof(long)))))            \
         = { __setup_str_##unique_id, fn, early }


#define __setup(str, fn)                                     \
   __setup_param(str, fn, fn, 0)
```

__setup

`__setup("console=", console_setup);`

- Expands to

```
static const char __setup_str_console_setup[] __initconst \
__aligned(1) = "console=";
static struct obs_kernel_param __setup_console_setup __used \
__section(.init.setup) __attribute__
((aligned((sizeof(long))))) \
= { __setup_str_console_setup, console_setup, early};
```

- Which expands to

```
static struct obs_kernel_param __setup_console_setup \
__section(.init.setup) = { __setup_str_console_setup,
console_setup, early};
```

- This stores the code in a table in section `.init.setup.`

On initialization...

- The table in `.init.setup` has
  - Parameter string (“`console=`“) and
  - Pointer to the function that processes it.
- This way the initialization code can process everything on the command line without knowing at compile time where all the code is.
- See section 5.3 for more details.