Porting Linux to a New ARM Platform

Deepak Saxena
dsaxena@plexity.net
Presented at Linux Bangalore 2004

Introduction/Goals

- ARM is very popular and growing
  - 75 percent of all 32-bit embedded CPUs [Wikipedia.org]
  - Growing usage in PDA, cell, and other mobile devices
- Not much info on Linux and ARM
  - Popular Kernel texts all cover core internals or drivers
  - No documentation on ARM-specific requirements
- Approach:
  - Go through kernel boot sequence
  - Explain places where platform-level hooks needed
  - Board bring-up debug

Assumptions

- Familiarity with ARM architecture
- Familiarity with HW architecture
- Basic understanding of kernel internals
- Familiarity with kernel build and config system

Step by Step

- Machine Registration
- Firmware to Kernel Transition
- Early kernel init
- Mapping I/O devices
- IRQ setup
- System timer tick
- Core subsystem initialization
- Final Board Initialization
- Device drivers

First Steps

- Register your machine type
  - Provides a unique numerical identifier for your machine
  - Provides a configuration variable for your machine
    - CONFIG_MACH_$MACHINE
  - Provides runtime machine-check
    - machine_is_xxx()
- http://www.arm.linux.org.uk/developer/machines/
- This information ends up in
  - *arch/arm/tools/mach-types*

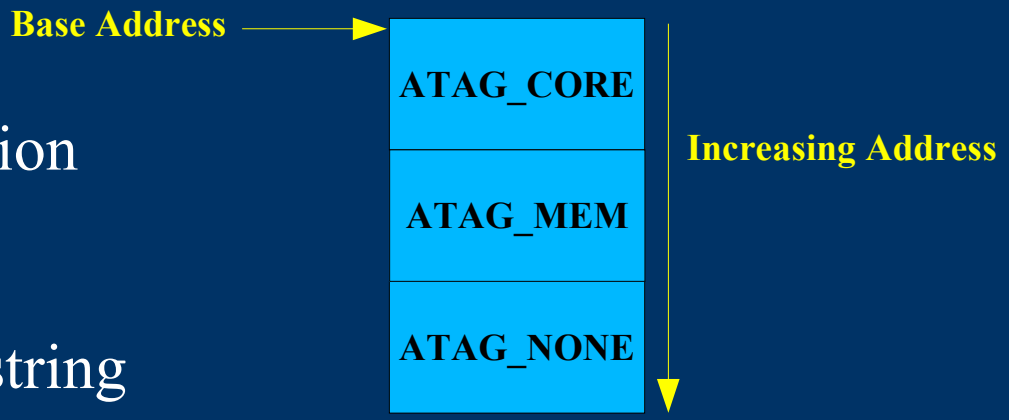
```
# machine_is_xxx        CONFIG_xxxx             MACH_TYPE_xxx           number
#
ebsa110                 MACH_EBSA110            EBSA110                 0
riscpc                  MACH_RPC                RISCPC                  1
nexuspci                MACH_NEXUSPCI           NEXUSPCI                3
ebsa285                 MACH_EBSA285            EBSA285                 4
netwinder               MACH_NETWINDER          NETWINDER               5
cats                    MACH_CATS               CATS                    6
```

Firmware Requirements

- x86 requires BIOS to perform certain tasks
- ARM Linux requires same from firmware
  - Detailed in *Documentation/ARM/Booting*
    - Initialize all memory controller and system RAM
    - Initialize a single serial port for early boot messages
    - Disable MMU
    - Disable all caches
    - Quiesce all DMA capable devices
    - Provide kernel parameter ATAG list
    - Required registers:
      - r0 = 0
      - r1 = machine number
      - r2 = &(ATAG list)

ATAG Parameter List

- Data structure for providing machine details to kernel
  - ATAG_CORE
  - ATAG_MEM
    - Memory size and location
    - One per memory bank
  - ATAG_CMDLINE
    - Kernel command line string
  - ATAG_NONE
    - Signifies end of parameter list
- Usually located in first 16KiB of RAM
  - Most common is @ RAM base + 0x0100
  - Must not be overwritten by decompresser or initrd
- ATAG defined in *include/asm-arm/setup.h*



Directory/File Structure

- *arch/arm/*
  - *mm*
    - Cache/TLB/page fault/DMA handling
  - *kernel*
    - core kernel setup, APIs, and syscall handling
  - *lib*
    - low-level helper functions (mostly ASM)
  - *common*
    - code shared across various machine types
  - *arch-$MACHINE*
    - Machine-type specific code (arch-ixp425, -pxa, -omap, etc)
  - *configs/$PLATFORM_defconfig*
    - Default configuration for $PLATFORM (lubbock, ipaq, etc)
- *include/asm-arm/arch-$MACHINE (include/asm/arch)*
  - Machine-specific headers

Early Init

- Early serial during decompression
  - arch_decomp_setup()
  - putstr()
  - *include/asm-arm/arch-$MACHINE/uncompress.h*
- Debug Serial Output
  - addruart, rx
    - Provide UART address in \rx
  - senduart rd, rx
    - Send character in \rd (@ address \rx)
  - busyuart rd, rx
    - Wait until UART is done sending
  - waituart rd, rx
    - Wait for Clear to Send
  - Found in *arch/arm/kernel/debug.S*

CPU Detection

- Large number of ARM CPU variants in production
  - Each on has different methods of managing cache, TLB
  - Sometimes need to build kernel to boot on various CPUs
- Kernel contains table of CPUs it supports
  - *arch/arm/mm/proc-$CPUTYPE*
  - *include/asm-arm/procinfo.h*
  - First thing kernel does is check CPUID with table of CPUs
  - Table contains a mask and expected value
    - If (cpuid & cpu_mask) == cpu_val we are OK
    - Otherwise we can't run on this CPU
  - If OK, call CPU's _setup function

Low Level CPU APIs

- Processor-specific functions
  - Data abort, CPU init, reset, shutdown, idle
  - *include/asm-arm/cpu-multi32.h*
- TLB handling
  - Flush user and kernel TLB entries
  - *include/asm-arm/tlbflush.h*
- Cache functions
  - Flush and clean kernel and user range from cache
  - Sync icache and dcache for new text
  - Sync dcache with memory for DMA operations
  - *include/asm-arm/cacheflush.h*
- User data functions
  - Clear and copy user page
  - *include/asm-arm/page.h*

Machine Detection

- If CPU is detected OK, check machine type
- Each platform has a machine descriptor structure:

```
MACHINE_START(IXDP425, "Intel IXDP425 Development
  Platform")
        MAINTAINER("MontaVista Software, Inc.")
        BOOT_MEM(PHYS_OFFSET,
  IXP4XX_PERIPHERAL_BASE_PHYS,
                IXP4XX_PERIPHERAL_BASE_VIRT)
        MAPIO(ixdp425_map_io)
        INITIRQ(ixp4xx_init_irq)
        .timer          = &ixp4xx_timer,
        BOOT_PARAMS(0x0100)
        INIT_MACHINE(ixdp425_init)
MACHINE_END
```

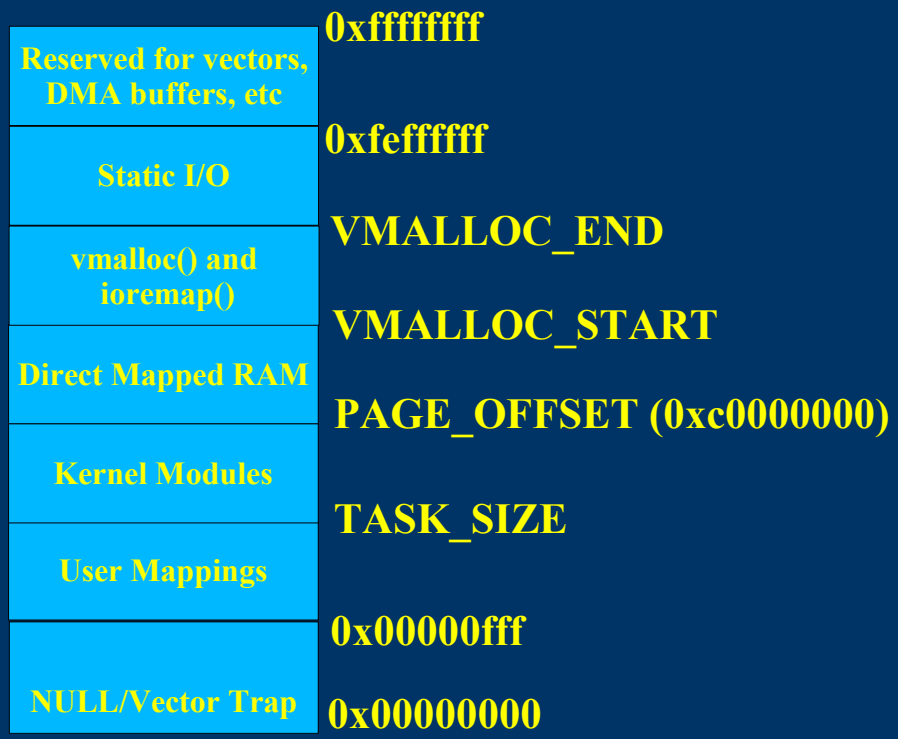
- Machine name/number from *arch/arm/tool/mach-types*

Static I/O Mapping

- Certain devices needed before VM is fully up and running
  - Interrupt controllers, timer tick
- Certain devices require large VM areas
  - Static mapping allows usage of 1MB sections
  - ioremap() uses only 4K pages
    - Larger TLB footprint
- Call mdesc->map_io()
- Call create_mapping() with static I/O mappings

ARM-Linux Memory Map

- Kernel memory is in upper 1GB
- Static mappings fit in VMALLOC_END – 0xefffffff
- VMALLOC_END is defined by you
  - *include/asm-arm/arch-$MACHINE/vmalloc.h*



ARM IRQs

- NR_IRQS defined in *include/asm-arm/arch/irqs.h*
- IRQ numbering is up to developer
- First level IRQ decoding done in ASM
  - *include/asm/arch-$MACHINE/entry-macro.S*
  - get_irqnr_and_base irqnr, irqstat, base, tmp
    - Linux IRQ number returned in \irqnr
    - Others are for temp calculations
- IRQ “Chips”
  - Chip defines a mask, unmask, and ack functions

System Timer Tick

- Initialized after IRQs
  - Load timer with LATCH
    - ((CLOCK_TICK_RATE + HZ/2) / HZ)
    - CLOCK_TICK_RATE is HW clock freq.
  - Request timer interrupt
    - Set SA_INTERRUPT
  - Timer ticks every (1/HZ)s
- Timer interrupt:
  - Call timer_tick()
  - Reload timer source if needed
- gettimeoffset() function
  - Returns number of *u*sec since last timer tick

Board Level Device Initialization

- After core subsystems are initialized, board_init() is called
  - Do any last needed fixups for the platform
  - Add platform devices (flash, I2C, etc)
  - Very board/platform specific

Device Drivers

- Live outside of arch/arm

Early Debug

- Sometimes need to debug before console is initialized
  - printascii() and friends
  - JTAG (Abatron BDI2000)

Working With the Community

- Plan on getting things upstream
  - Decreases your workload
- Release early/release often
  - Work with the community to fix issues
- Do not make core changes or create new APIs in the dark
  - Wastes time fo everyone involved
- ARM Linux Website
  - http://www.arm.linux.org.uk

Questions?

Backup Material

ARM PCI Support