EASY ARM

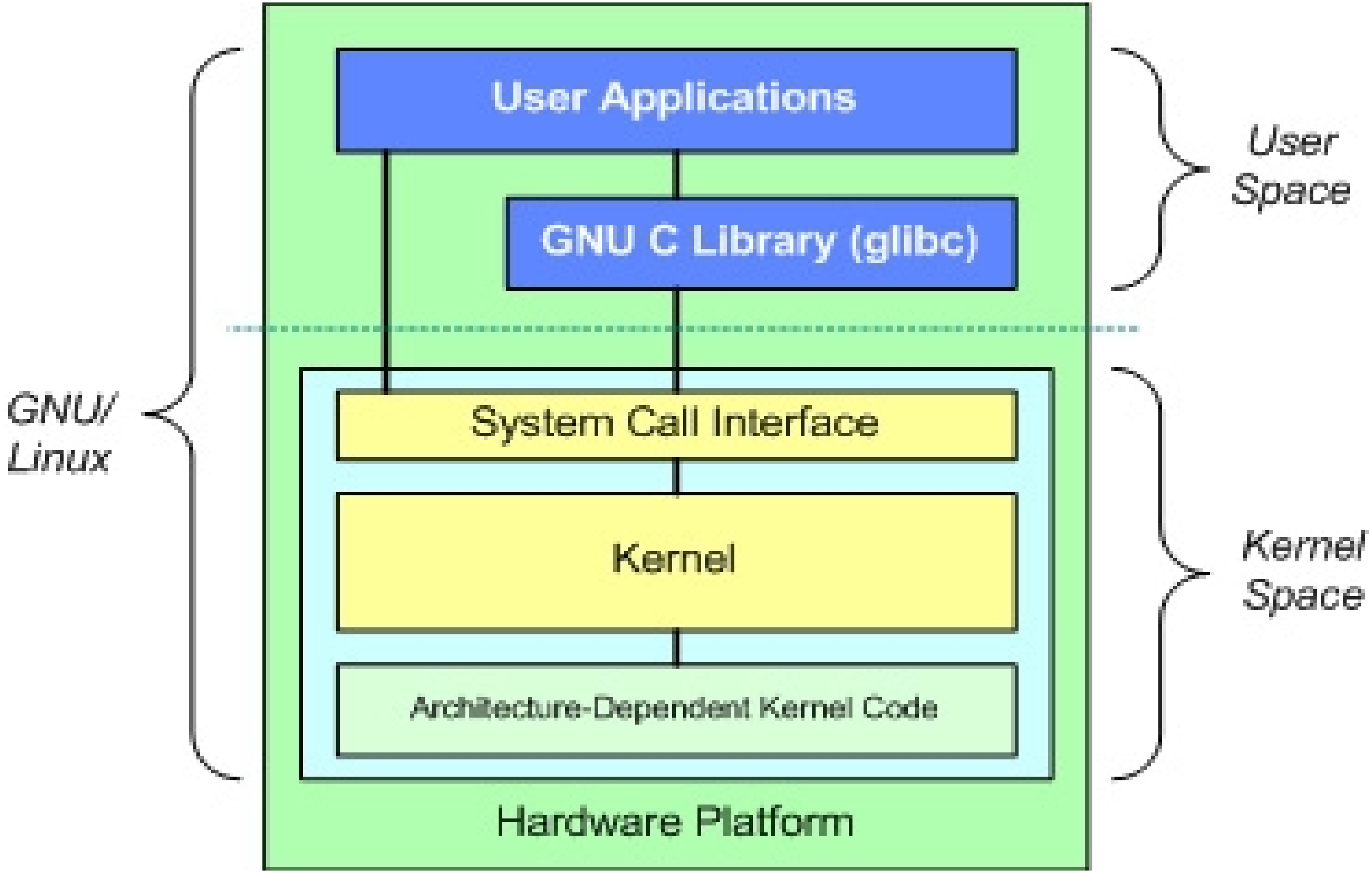
Embedded Linux Workshop on Blueboard-AT91

B. Vasu Dev

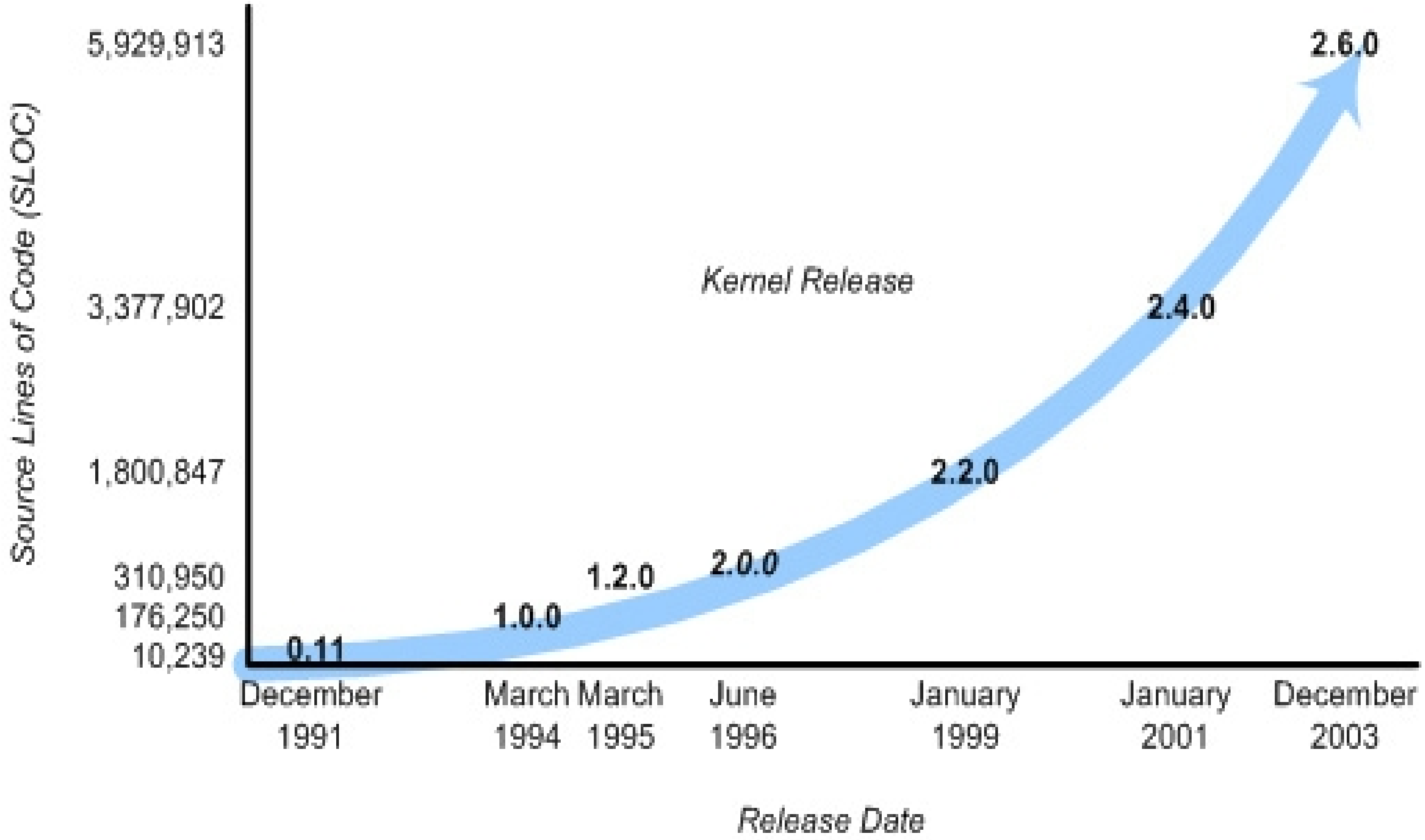
vasu@easyarm.com

Linux Kernel

Linux OS Architecutre



Major Linux kernel releases



Kernel

Kernel is a resource managare, where resource can be a process, memory or hardware.

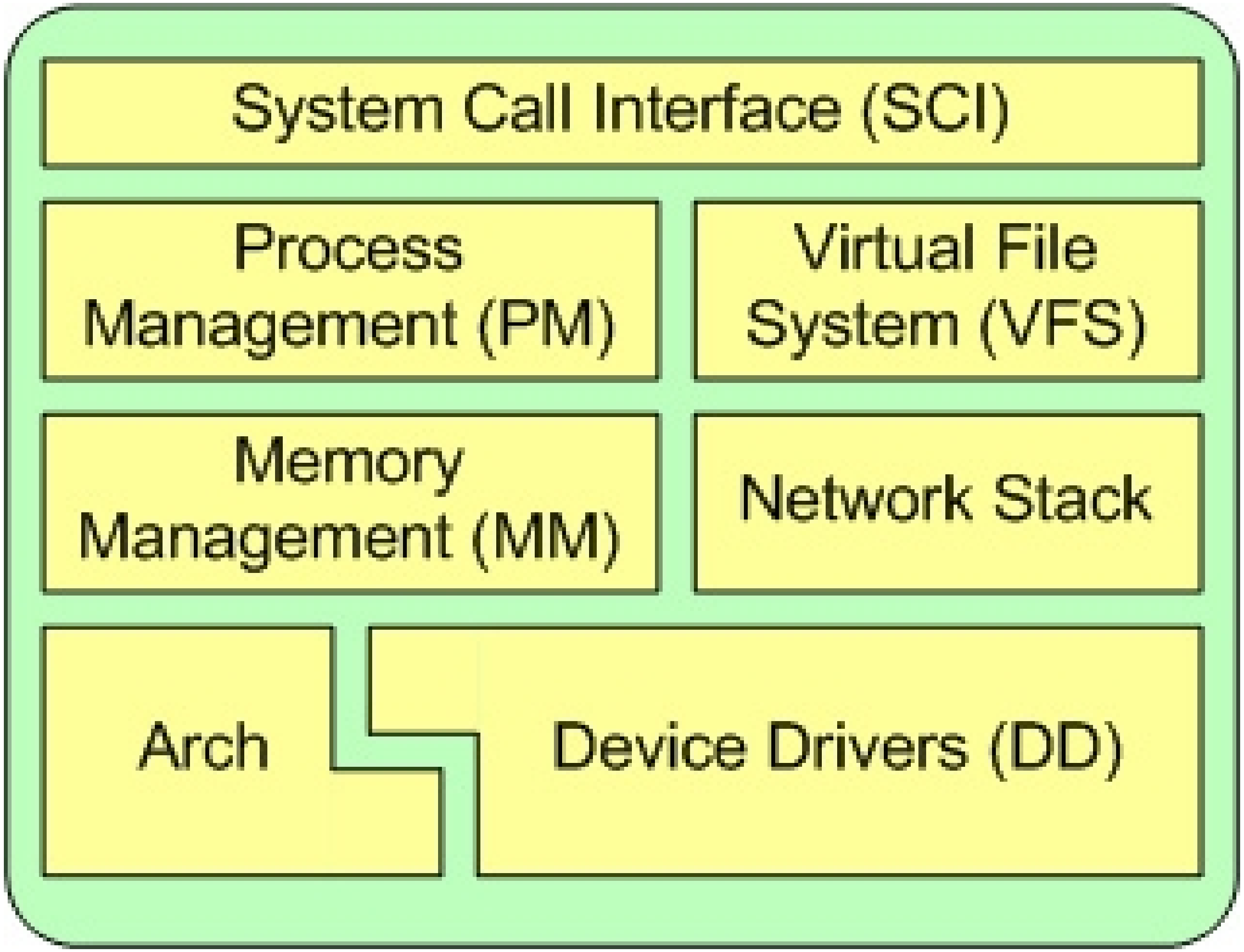
The kernel manages and arbitrates access to the resource between multiple users.

The Kernel is the core of the Operating System.

Collection of routines mostely written in C and few of them in Assembly language which communicated with hardware.

It is the part of the Linux System that is loaded into memory when the system is booted.

Kernel Subsytems



Kernel Subsystems

System Call Interface

The SCI is a thin layer that provides the means to perform function calls from user space into the kernel.

A system call is an interface between a user-space application and a service that the kernel provides

Process Management

The Process Management system controls the creation, termination, accounting, and scheduling of processes.

It also facilitates and manages the complex task of the creation of child processes.

Memory Managment

Manages the physical memory in terms of pages (4Kb Buffer). Provides the mechanism of swaping, where the pages are swaped from phy. memory to hard disk.

Kernel Subsytems

Virtual File System

Provides a common interface abstraction for file systems. The VFS provides a switching layer between the SCI and the file systems supported by the kernel. At the top of the VFS is a common API abstraction of functions such as open, close, read, and write.

Network Stack

Provides the wide support of network protocol such as IP, TCP, UDP and raw socket access API's.

Device Drivers

A set of routines needed to operate a specific device is known as device driver.

They hide completely the details of how the device works. The linux device drivers framework makes easy to port & add new hardware without effecting the top layer.

Architecture Dependent Code

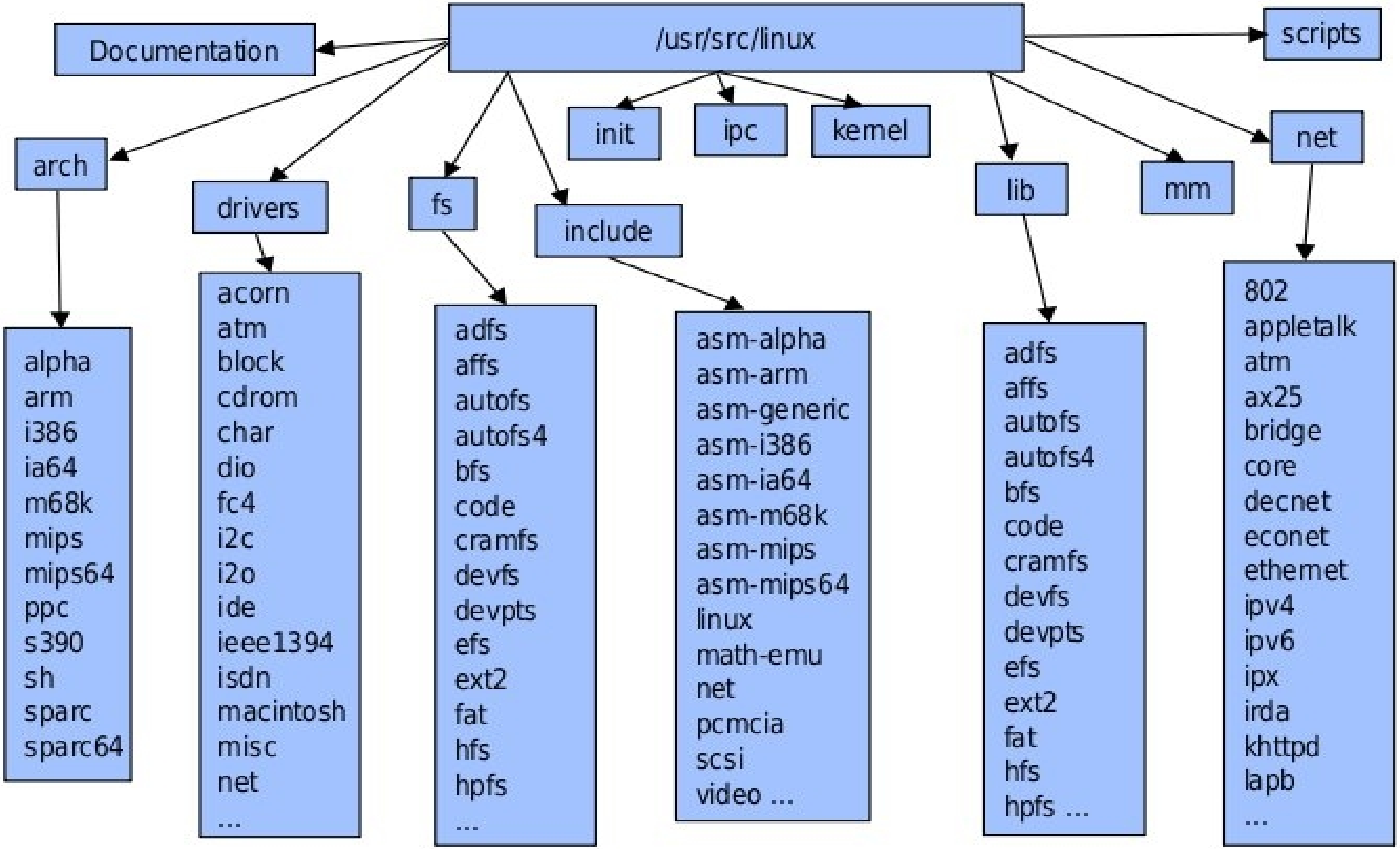
The code wich makes linux to run on differnet hardware like x86, ARM, MIPS, PowerPC etc. Most of the BSP code resides here.

Kernel Subsytems

Take a simple senario of runing “Hello World” Program and identifiy where all these kernel subsytems works.

1. Where do the hello.bin executes in memory.
2. If same hello.bin is executed in 2 different terminal who manages
3. How does your program is able to write on console

Kernel Source



Kernel Source

Top Level Files : Build System & General Info

Makefile, kbuild, readme, lisence, maintainers & reporting-bugs.

linux-x.x.xx/Documentaion : Your first frinend to teach you lot about linux. Contains many subdirectories and files documented on specific topic and general documentaion.

linux-x.x.xx/arch : Hardware dependent code for different architecuters like x86, arm, mips, avr, powerpc ..... .

Each port directory (ex. arm) contains subdirectories **boot, lib, mm, kernel...** which overrides the code stub in architecture independent code.

- **boot** - assembly files, related to booting Linux
- **lib** - optimized routines for common tasks (e.g. memcpy)
- **mm** - i386 specific memory management
- **kernel** - the bulk of the i386 code, including IRQ handling, processes, signals and pci support, to name a few areas

Kernel Source

linux-x.x.xx/drivers : The huge code repository of the drivers supported by linux.

➔ net/, sound/, usb/, atm/, ide/, scsi/, etc, etc

➔ character drivers lives in /drivers/char and the block drivers in /drivers/block.

➔ The most experimental location for you while doing porting on a new board.

linux-x.x.xx/fs : Linux supports varity of file systems few of them are ext2/3/4, fat, jfs, jffs2, squashfs, nfs, ntfs and many more. Code related to each file system resides in their subdirectories.

VFS is the manager who deals with all these file system and provide a common interface to the user level.

linux-x.x.xx/include : header files live here

➔ asm-* include architecture specific header files (complement arch/).

➔ The 'asm' symbolic link is created as part of the build process depending on which architecture we are compiling for.

➔ grep here first when looking for an API or a constant most important (relevant) header files live in include/linux

Kernel Source

linux-x.x.xx/kernel : The core kernel code. Some of which is

➔ sched.c – "the main kernel file":
scheduler, wait queues, timers, alarms, task queues.

➔ Process control:
fork.c, exec.c, signal.c, exit.c etc...

➔ Kernel module support:
kmod.c, ksyms.c, module.c.

➔ Other operations:
time.c, resource.c, dma.c, softirq.c, itimer.c.
printk.c, info.c, panic.c, sysctl.c, sys.c.

Kernel Source

linux-x.x.xx/mm : The memory managment unit.

Paging and swapping:

swap.c, swapfile.c (paging devices), swap_state.c (cache).
vmscan.c – paging policies, kswapd.
page_io.c – low-level page transfer.

Allocation and deallocation:

slab.c – slab allocator.
page_alloc.c – page-based allocator.
vmalloc.c – kernel virtual-memory allocator.

Memory mapping:

memory.c – paging, fault-handling, page table code.
filemap.c – file mapping.
mmap.c, mremap.c, mlock.c, mprotect.c.

Kernel Source

linux-x.x.xx/lib - generic library support routines

linux-x.x.xx/net - networking support, ipv4 and v6, tcp, other esoteric protocols

linux-x.x.xx/ipc - SYSV interprocess communications mechanisms

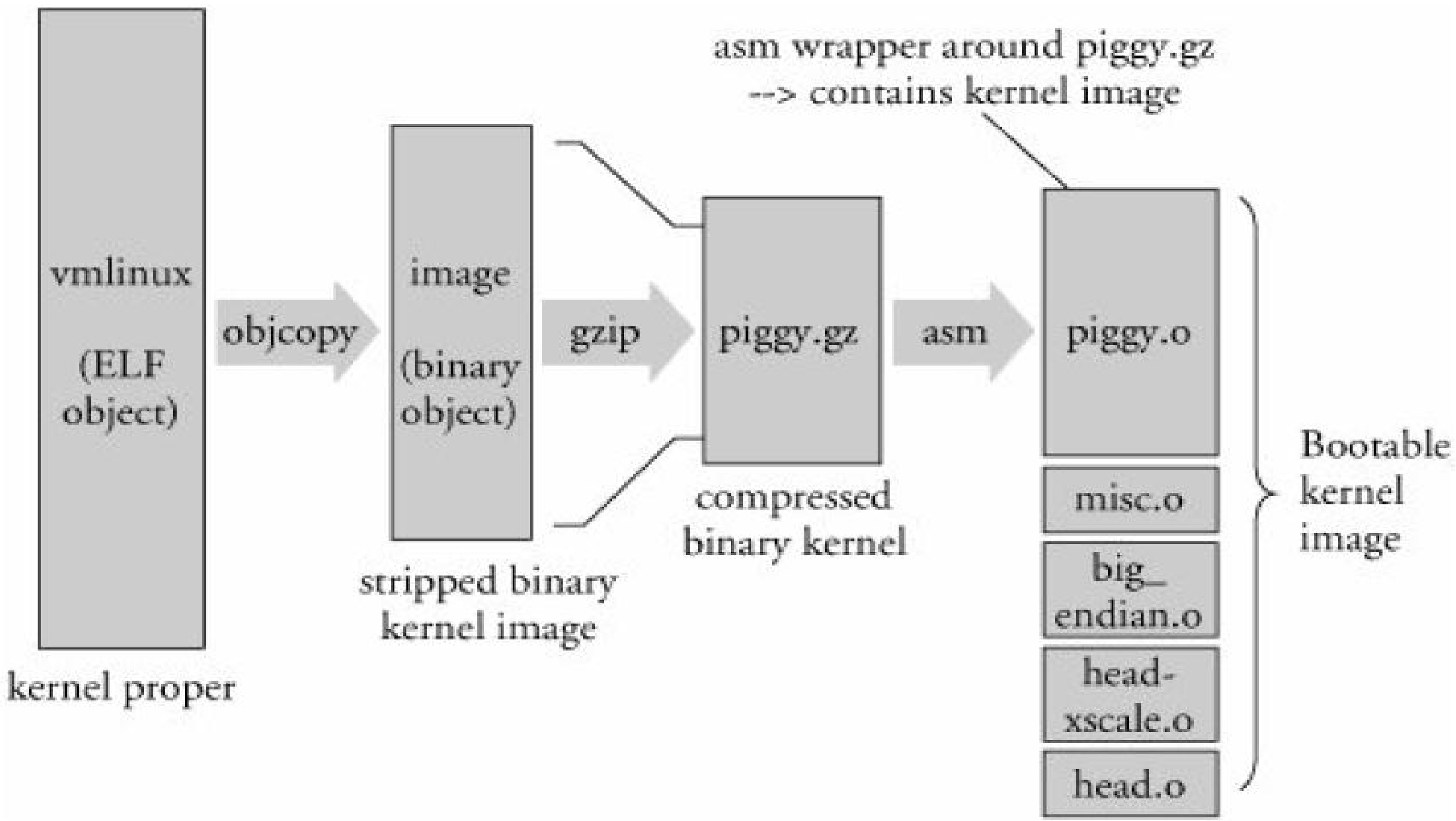
linux-x.x.xx/init - kernel initialization and startup

linux-x.x.xx/crypto (new addition) - cryptographic support

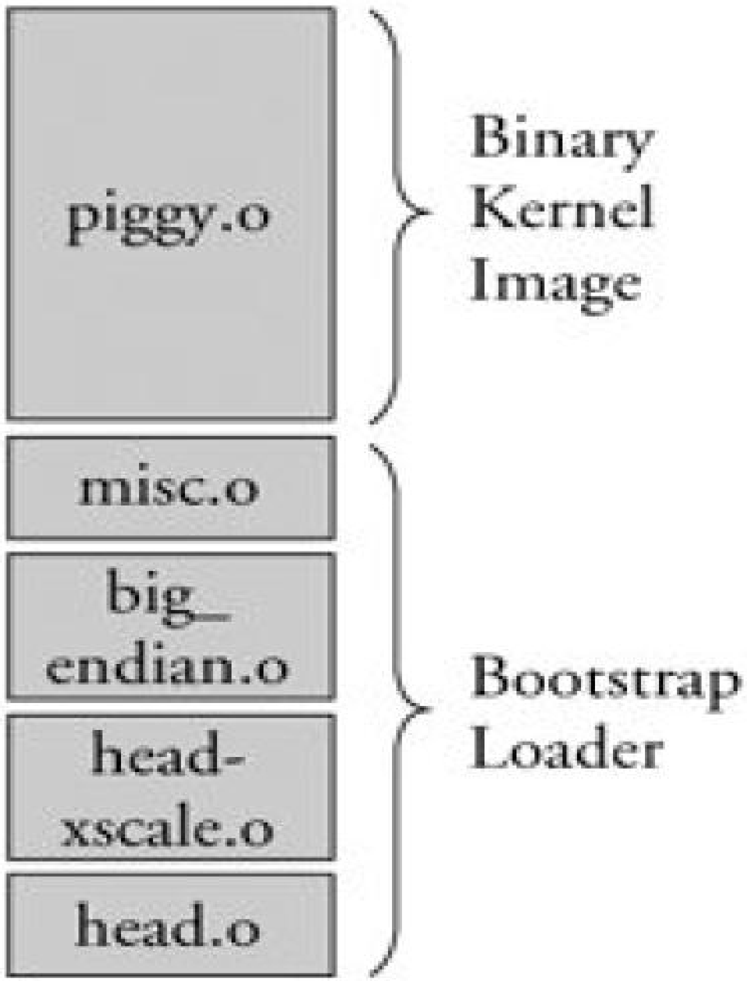
linux-x.x.xx/scripts - various scripts, some used for the build system

Kernel Image Components

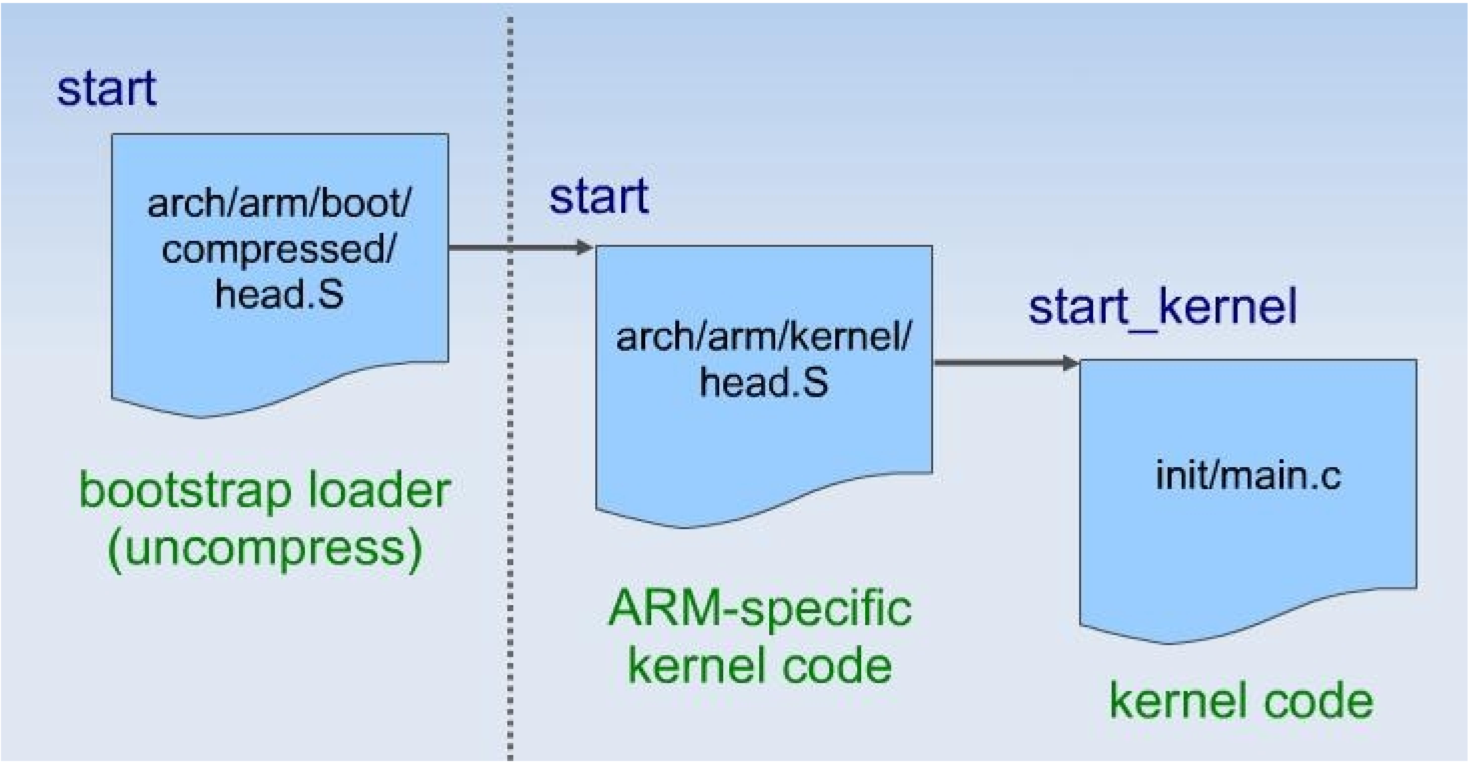
Composite kernel image construction



Kernel Image Components



Kernel Booting & Initialization



Kernel Booting & Initialization

Bootstrap -> U-Boot -> Kernel -> RootFS(Init)

U-boot:

_start (cpu/arm920t/start.S)
start_code (cpu/arm920t/start.S)
start_armboot (lib_arm/board.c)
board_init (board/kb9202/kb9202.c)
timer_init (cpu/arm920t/at91/timer.c)
serial_init (drivers/serial/at91rm9200_usart.c)
main_loop (lib_arm/board.c)

This gives you uboot command promt.

Kernel Booting & Initialization

Loading & Starting Kernel

If kernel image is loaded using loadb / tftp / copied from flash memory to RAM and bootm command executed.

do_bootm (common/cmd_bootm.c)
bootm_start (common/cmd_bootm.c)
bootm_load_os (common/cmd_bootm.c)
do_bootm_linux (lib_arm/bootm.c)
stext (linux/arch/arm/kernel/head.S)

Control is given to linux.

Kernel Booting & Initialization

U-Boot does this:

Configure the memory system.
Load the kernel image at the correct memory address.
Optionally load an initial RAM disk at the correct memory address.
Initialise the boot parameters to pass to the kernel.
Obtain the ARM Linux machine type
Enter the kernel with the appropriate register values

Control is given to linux.

Kernel Booting & Initialization

(1) Kernel Invocation Process

(a) zImage Entry Point
(b) PERFORM BASIC HARDWARE SET UP
(c) PERFORM BASIC ENVIRONMENT SET UP (stack etc)
(d) CLEAR BSS

[Now We have set up the run time environment for the code to be executed next]

(e) DECOMPRESS THE KERNEL IMAGE
(f) Execute the decompressed Kernel Image
- INITIALIZE PAGE TABLES
- ENABLE MMU
- DETECT CPU (& optoinal FPU) TYPE & SAVE THIS INFO

[With above set up, we are now ready to execute a general C Code. Till now we only executed asm routines.]

(g) The First Kernel C function
- DO FURTHER INITIALIZATIONS
- LOAD INITRD

[The above code is being executed by swapper process, the one with pid 0]

(h) The Init Process
- FORK INIT PROCESS
- Init process is with pid 1
- Invoke Scheduler
- RELINQUISH CONTROL TO SCHEDULER

Kernel Booting & Initialization

Linux Kernel (1):

stext (arch/arm/kernel/head.S:78)

__lookup_processor_type (arch/arm/kernel/head-common.S:160)

__lookup_machine_type (arch/arm/kernel/head-common.S:211)

__create_page_tables (arch/arm/kernel/head.S:219)

__arm920_setup (arch/arm/mm/proc-arm920.S:389)

__enable_mmu (arch/arm/kernel/head.S:160)

__turn_mmu_on (arch/arm/kernel/head.S:205)

__switch_data (arch/arm/kernel/head-common.S:20)

start_kernel (init/main.c:529)

Kernel Booting & Initialization

Linux Kernel (2)

start_kernel (init/main.c:529)
tick_init(kernel/time/tick-common.c:413)
setup_arch (arch/arm/kernel/setup.c:666)
setup_machine (arch/arm/kernel/setup.c:369)
lookup_machine_type ( )
setup_command_line (init/main.c:408)
build_all_zonelists (mm/page_alloc.c:3031)
parse_args (kernel/params.c:129)
mm_init (init/main.c:516)
mem_init (arch/arm/mm/init.c:528)
kmem_cache_init (mm/slab.c, mm/slob.c, mm/slub.c)
sched_init (kernel/sched.c)

Linux Kernel (3)

init_IRQ (arch/arm/kernel/irq.c)
init_timers (kernel/timer.c:1713)
hrtimers_init (kernel/hrtimer.c:1741)
softirq_init (kernel/softirq.c:674)
console_init (drivers/char/tty_io.c:3084)
vfs_caches_init (fs/dcache.c:2352)
mnt_init (fs/namespace.c:2308)
init_rootfs ()
init_mount_tree (fs/namespace.c:2285)
do_kern_mount (fs/namespace.c:1053)
set_fs_pwd(fs/fs_struct.c:29)
set_fs_root(fs/fs_struct.c:12)

Linux Kernel (4)

bdev_cache_init (fs/block_dev.c:465)
chrdev_init (fs/char_dev.c:566)
signals__init (kernel/signal.c:2737)
rest_init (init/main.c:425)
kernel_thread (431, arch/arm/kernel/process.c:388)
kernel_thread() creates a kernel thread and control is given to kernel_init().
kernel_init (431, init/main.c:856)
do_basic_setup (888, init/main.c:787)
init_workqueues (789, kernel/workqueue.c:1204)
driver_init (793, drivers/base/init.c:20)
do_initcalls (796, init/main.c:769) /* Calls all subsytems init functions */
prepare_namespace (906, init/do_mounts.c:366)
initrd_load (399, init/do_mounts_initrd.c:107)

Linux Kernel (5)

rd_load_image (117, init/do_mounts_rd.c:158) /* if initrd is given */
identify_ramdisk_image (179, init/do_mounts_rd.c:53)
handle_initrd (119, init/do_mounts_initrd.c:37) /*if rd_load_image is success */
mount_block_root (45, init/do_mounts.c:233)
do_mount_root (247, init/do_mounts.:218)
mount_root (417, init/do_mounts.c:334) /* if initrd not given */
mount_block_root (359, init/do_mounts.c:233)
do_mount_root (247, init/do_mounts.c:218)
init_post (915, init/main.c:816)
run_init_process (847, init/main.c:807)
kernel_execve (810, arch/arm/kernel/sys_arm.c:81)
|
User Space |
————- |
init() /*userspace /sbin/init */

Kernel Configuration

Many ways to invoke kernel configuration

make <config_rule>

config_rules:

config	Update current config using a line-oriented program
menuconfig	Update current config using a menu-based program
xconfig	Update current config using a QT-based front end
gconfig	Update current config using a GTK-based front end
oldconfig	Update current config using a provided .config as the base
randconfig	New config with random answer to all options
defconfig	New config with default answer to all options
allmodconfig	New config that selects modules, when possible
allyesconfig	New config in which all options are accepted with yes
allnoconfig	New minimal config

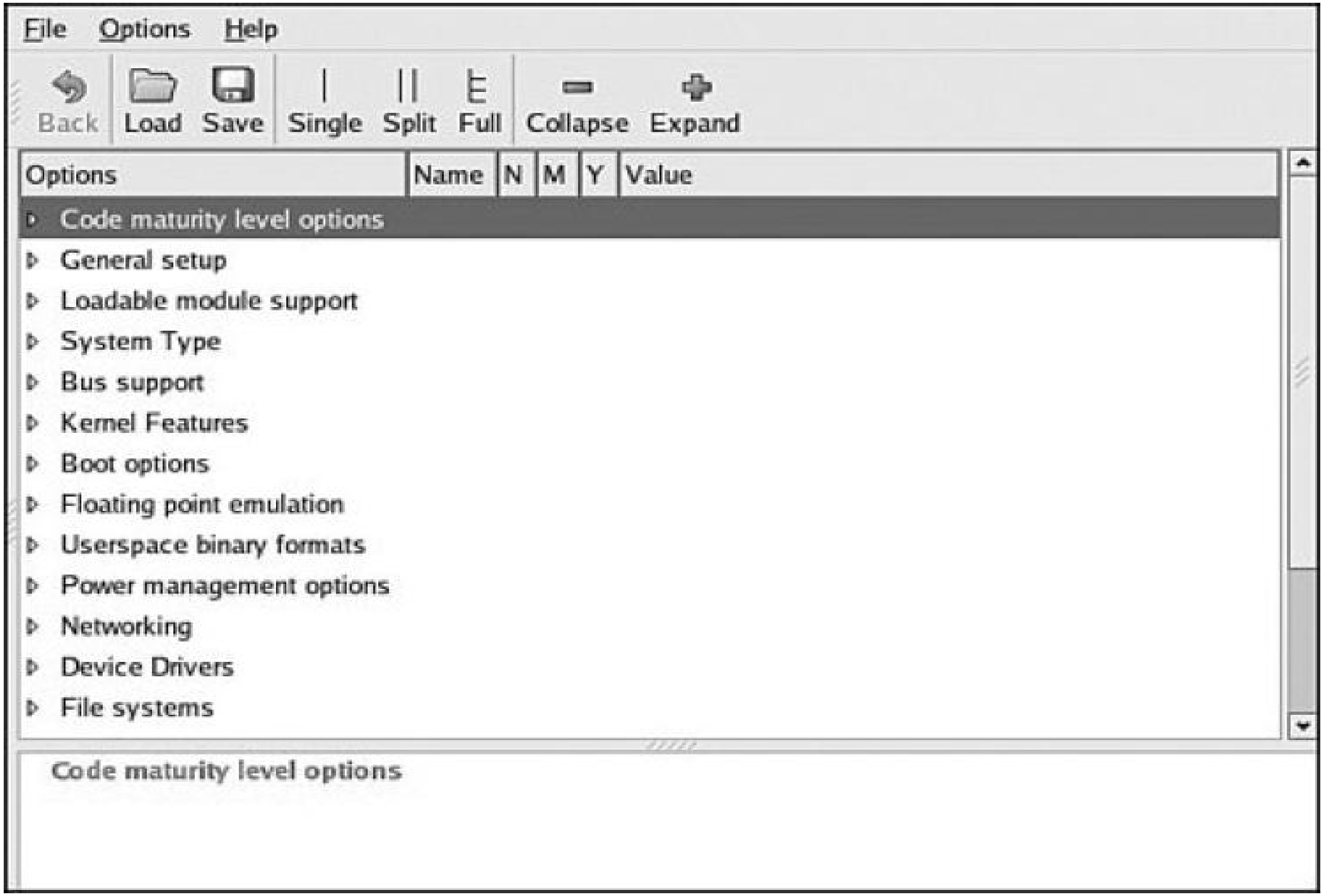
Kernel Configuration

Most Commonly used one   make menuconfig

```
Linux Kernel v2.6.11.6 Configuration

                         Linux Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

          Code maturity level options  --->
          General setup  --->
          Loadable module support  --->
          Processor type and features  --->
          Power management options (ACPI, APM)  --->
          Bus options (PCI, PCMCIA, EISA, MCA, ISA)  --->
          Executable file formats  --->
          Device Drivers  --->
          File systems  --->
          Profiling support  --->
          Kernel hacking  --->
        ↓(+)

              <Select>    < Exit >    < Help >
```

Kernel Configuration

Graphical based (Needs GTK) make xconfig



Kernel Config& Compilation

Board Default Configuration

make <boardname_defconfig>

make ecbat91_defconfig // For bb_at91

make ARCH=<arch> CROSS_COMPIE= <crosstool-prefix> <Image_type>

make ARCH=arm CROSS_COMPILE=arm-softfloat-linux-gnu- uImage

Uploading Kernel Image

Uploading the Ketnel Image to the target is board specific.

1. Serial Port
2. TFTP
3. USB
4. SD-CARD

Bootargs to Kernel

Bootargs provide the configurable information to the kernel by the bootloader.

1. Console
2. RAM Size
3. RootFS Source
4. NFS Information if configured
5. Many more……….

Porting Kernel to New Board

Step by Step

- Machine Registration
- Adding New board code
- Modifying Configuration files
- Modifying Make files

Porting Kernel to New Board

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*

Porting Kernel to New Board

arch/
    arm/
        machat91/

AT91 generic code
clock.c, leds.c, irq.c, pm.c

CPU specific code for the AT91RM900
at91rm9200.c, at91rm9200_time.c,
at91rm9200_devices.c

Board specific code
board-ecbat91.c

Porting Kernel to New Board

The at91rm9200_devices.c file doesn't implement the drivers for the platform devices

The drivers are implemented at different places of the kernel tree

For the bb_at91/ecb_at91 board

- USB host, driver at91_ohci, drivers/usb/host/ohciat91.c
- USB device, driver at91_udc, drivers/usb/gadget/at91_udc.c
- Ethernet, driver macb, drivers/net/macb.c
- NAND, driver atmel_nand, drivers/mtd/nand/atmel_nand.c
- I2C on GPIO, driver i2cgpio, drivers/i2c/busses/i2cgpio.c
- SPI, driver atmel_spi, drivers/spi/atmel_spi.c
- Buttons, driver gpiokeys,
- drivers/input/keyboard/gpio_keys.c

All these drivers are selected by the readymade configuration file

Porting Kernel to New Board

Configuration file

A configuration option must be defined for the board, in arch/arm/machat91/Kconfig

```
config MACH_ECBAT91
	bool "emQbit ECB_AT91-V1 Single Board Computer"
	depends on ARCH_AT91RM9200
	help
	  Select this if you are using emQbit's ECB_AT91 (V1) Single Board Computer.
	  The ECB_AT91 (V1) is Open Hardware.
	  <http://wiki.emqbit.com/products>
```

This option must depend on the CPU type option corresponding to the CPU used in the board. Here the option is ARCH_AT91RM9200, defined in the same file.

A default configuration file for the board can optionally be stored in arch/arm/configs/. For our board, it's ecbat91_defconfig

Porting Kernel to New Board

Make File Changes

The source files corresponding to the board support must be associated with the configuration option of the board

This is done in arch/arm/machat91/Makefile

```
obj-$(CONFIG_MACH_ECBAT91)  += board-ecbat91.o
```

Porting Kernel to New Board

The board-ecbat91.c implements and fulfills the Machine Structure

```
MACHINE_START(ECBAT91, "emQbit's ECB_AT91 V1")
	/* Maintainer: Emqbit.com */
	.phys_io = AT91_BASE_SYS,
	.io_pg_offst	= (AT91_VA_BASE_SYS >> 18) & 0xfffc,
	.boot_params= AT91_SDRAM_BASE + 0x100,
	.timer		= &at91rm9200_timer,
	.map_io		= ecb_at91map_io,
	.init_irq	= ecb_at91init_irq,
	.init_machine = ecb_at91board_init,
MACHINE_END
```

RootFs

The minimal rootfs:

```
.
|-- bin
|    |-- busybox
|    |-- echo -> busybox
|    |-- mount -> busybox
|    '-- sh -> busybox
|-- dev
|    |-- console
|    |-- ram0
|    '-- ttyS0
|-- etc
|-- linuxrc
'-- proc

4 directories, 8 files
```

RootFs

Components of Root File System

1. Unix Directory Structure
2. Device File / Nodes
3. Scripts
4. Binary files for commands or Busybox
5. Libraries

RootFs

Compiling Busybox

Hands-On

?