Salon RTS 2010
Journée spéciale « Linux embarqué »

LINAGORA

# Portage kernel sur une architecture ARM

*retour d'expérience sur une aventure débugante*

Gilles BLANC
Expert Linux embarqué
Groupe LINAGORA
gblanc@linagora.com
http://gblanc.blogs.linagora.com/
www.gillesblanc.com

# Licence

## By – Non commercial France

You are granted freedom to :

- Reproduce, distribute, and publish this work,
- Modify this document.

Provided the following conditions hold :

Author. The initial author must be named.


NonCommercial. You may not use this work for commercial purposes.

For any reuse or distribution, you must make clear to others the license terms of this work.

Any of the above conditions can be waived if you get permission from the copyright holder.

In no way are any of the following rights affected by the license: your fair dealing or fair use rights, or other applicable copyright exceptions and limitations, the author's moral rights, and rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

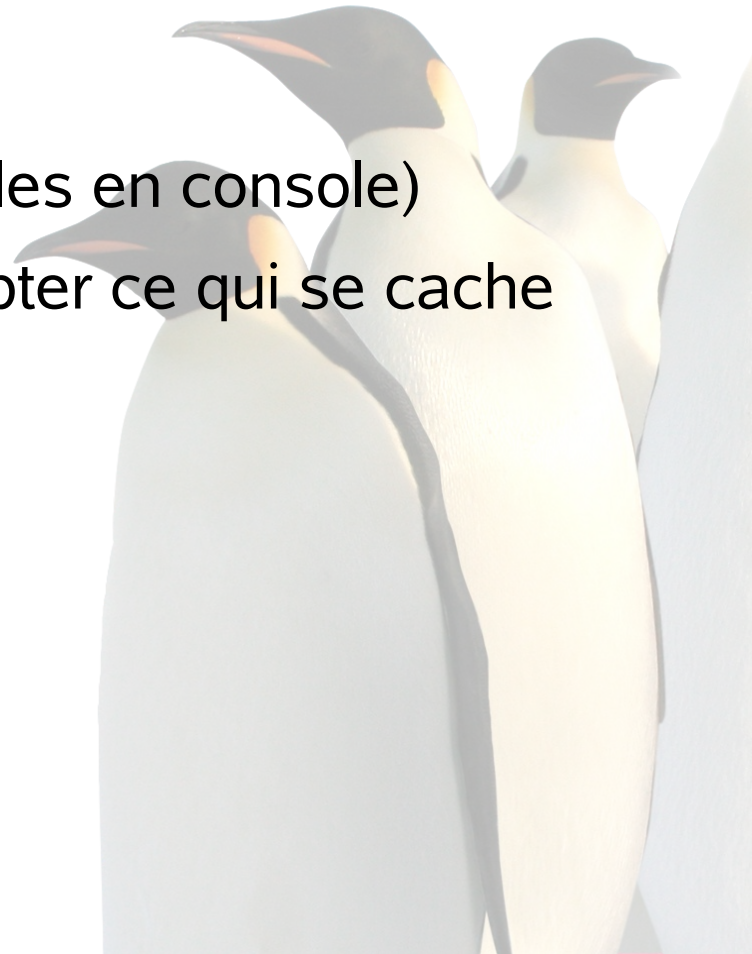
- Problématique
- Environnement de dev/debug libre bas niveau
- Démarrer un noyau Linux depuis la RAM via J-Tag
- Débugger le kernel pour un portage

- Au programme :
  - Des frayeurs
  - Des techniques Jedi©
  - Des trucs et astuces dignes de McGyver©
  - Un mal de crâne
- Pas au programme :
  - Un SDK type « Commercial, off-the-shelft »
  - De l'auto-magique
  - Du commercial

# La problématique et le bug



- Carte sur processeur ARM (Embest SBC2440), peu chère
  - Noyau Linux disponible
  - Le fondeur (chinois) nous assure la compatibilité
  - Le noyau fourni est trop vieux (2.6.13), on veut du 2.6.25/2.6.27 (2008)
- Un noyau nouveau ne passe pas
  - Boot ésotérique (caractères illisibles en console)
  - Réactivité assez nulle (sans compter ce qui se cache derrière)
- Des moyens (très) limités
  - Hors cadre projet
  - Interlocuteurs aussi embarrassés
  - Câble J-Tag peu cher...



# 1er problème : booter un nouveau kernel



- Doc officielle très parcellaire
- Indiqué d'utiliser le « wriggler » pour flasher en RAM :
  - (très) long (passe par le port parallèle)
  - Bourrin (reflasher à chaque fois ⁉)
  - L'expérience dit le contraire
- Décision de passer à OpenOCD :
  - Projet de gestion de J-Tag libre
  - Éthique et gratuit
- Mais :
  - Le CPU est-il supporté ?
  - Le câble J-TAG est-il supporté ?
  - La carte va-t-elle marcher ?

- Retours d'expérience
  - Ça a l'air rudement complexe (LPC sans MMU) :

http://www.cs.tau.ac.il/lin-club/lin-club_files/Ori-Idan-Linux-ARM7TDMI.odp

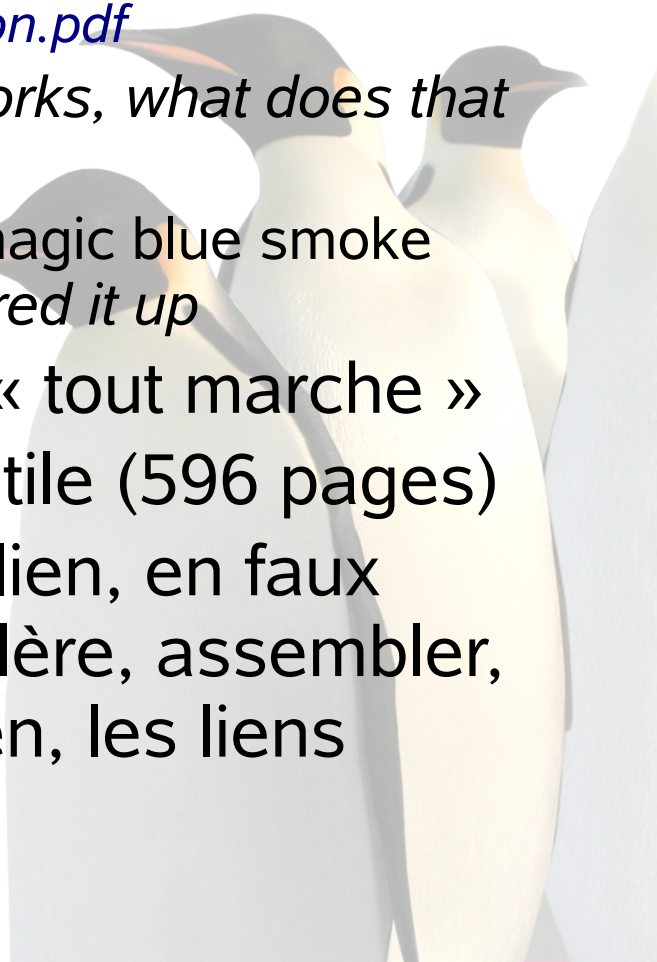
  - Et même plus (avec MMU) :

*http://linuxdevices.com/files/article078/JTAG-Anderson.pdf*

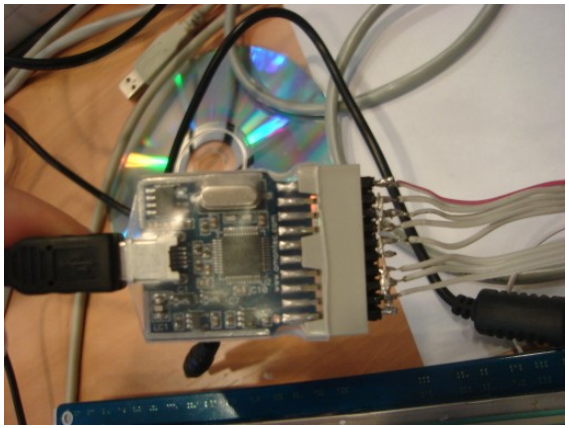
*When the hardware folks say the board works, what does that mean ?*

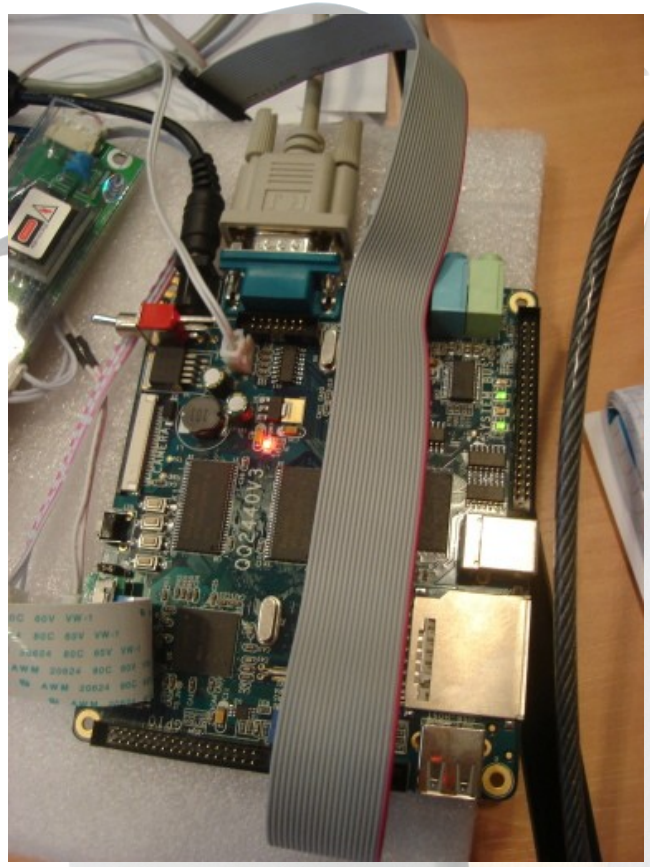
*-> Frequently, it simply means that the* magic blue smoke *doesn't escape the chips when they powered it up*

- Documentation Linux fournie en mode « tout marche »
- Recherche difficile de la « vraie » doc utile (596 pages)
- Autre documentation en Chinois, en Italien, en faux Anglais, pour un CPU coréen : vraie galère, assembler, recouper, etc ; Google n'y comprend rien, les liens changent sans cesse



- Carte Embest SBC2440
- Câble J-Tag Amontec :
  - J-TagKey (29€, autant en transport)
    http://www.amontec.com/jtagkey-tiny.shtml
  - Pas le bon espacement (2mm vs 2.5mm) : réadaptation manuelle des PINs
- CPU ARM9 S3C2440 (Samsung)
- OpenOCD version dev (SVN r888)



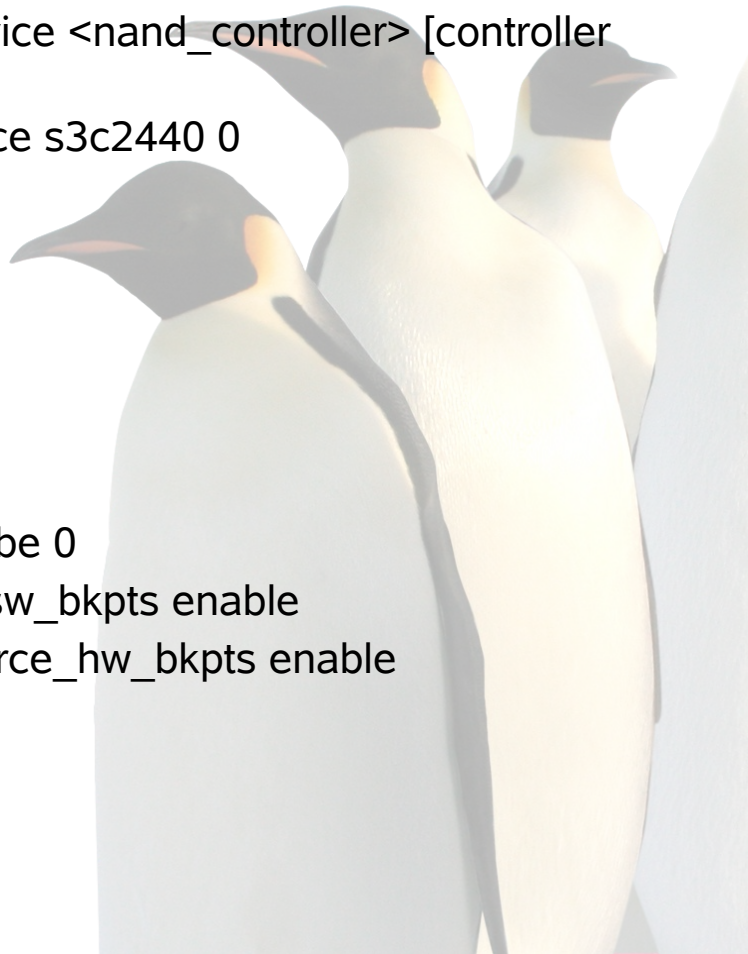


- http://openocd.berlios.de/web/
- Open On-Chip Debugger, « Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing »
- Projet libre de gestion de câbles J-Tag
- Très bien documenté
- Vivant (entraide communautaire aussi : ML, forums)

# Démarrer OpenOCD

- # openocd -f ./openocd-s3c2440.cfg
- Ne pas oublier de monter usbfs et la lib ftdi
- Fichier de conf :

```
interface ft2232
jtag_speed 0
jtag_ntrst_delay 100
jtag_nsrst_delay 100

ft2232_vid_pid 0x0403 0xcff8
ft2232_layout "jtagkey"
ft2232_device_desc "Amontec JTAGkey"

jtag_device 4 0x1 0xf 0xe

#daemon_startup attach
target arm920t little 0 arm920t

reset_config trst_and_srst combined

working_area 0 0x33F00000 0x4000 nobackup

#run_and_halt_time 0 500

#nand device <nand_controller> [controller options]
nand device s3c2440 0


#script
init
reset
halt
#nand probe 0
#arm7_9 sw_bkpts enable
arm7_9 force_hw_bkpts enable
```

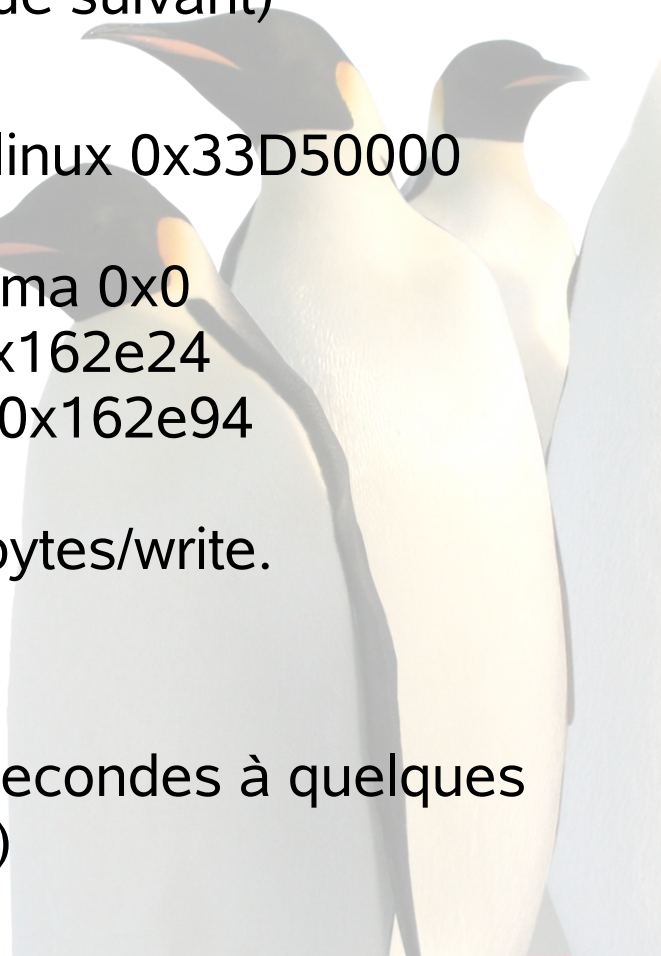


# Démarrer gdb

- Mode remote : permet de se connecter à un gdb-server distant (en l'occurrence OpenOCD) :
  - $ target remote localhost:3333
- Charger le kernel Linux en RAM :
  - Déterminer une adresse mémoire (cf slide suivant)
  - (gdb) load ./S3C2440/linux-2.6.25/arch/arm/boot/compressed/vmlinux 0x33D50000

    Loading section .text, size 0x162e24 lma 0x0
    Loading section .got, size 0x70 lma 0x162e24
    Loading section .got.plt, size 0xc lma 0x162e94
    Start address 0x0, load size 1453728
    Transfer rate: 55630 bits/sec, 15801 bytes/write.

    (gdb)
  - Temps dépend du J-Tag : de quelques secondes à quelques minutes (ici très long : câble peu cher)



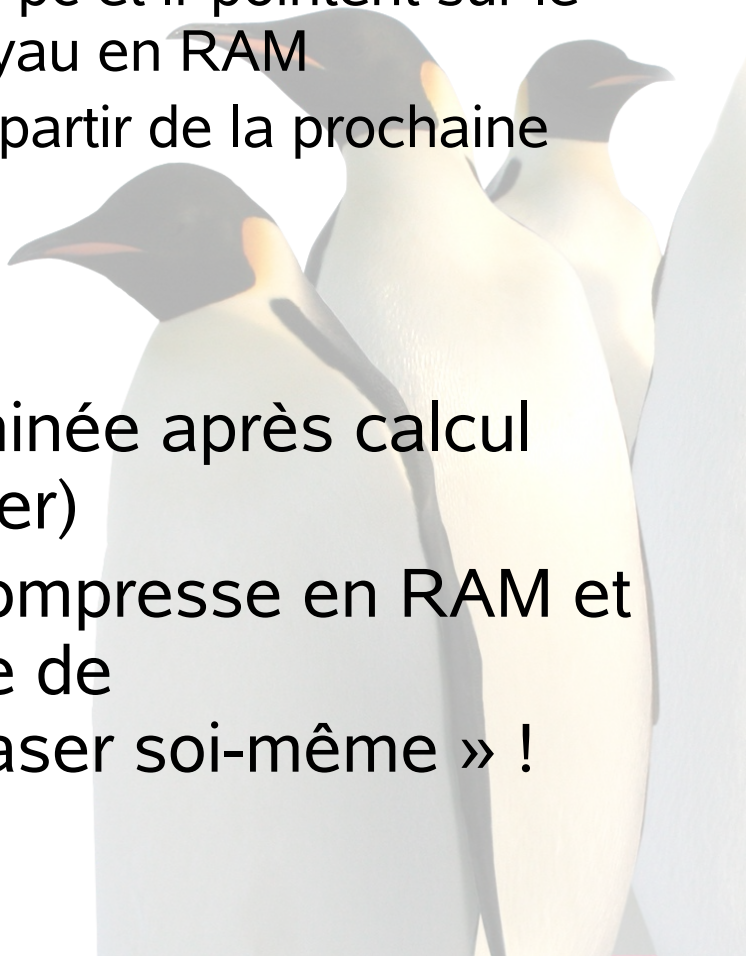
# Démarrer le kernel en RAM



- Commandes gdb :

```
(gdb) set $r0=0          <- registre r0 à 0 (convention)
(gdb) set $r1=362        <- r1 à 362 p/r à l'ID (arch/arm/tools/mach-types)
(gdb) set $r2=0          <- r2 à 0 pour se servir de la ligne de
                            commande compilée en option
(gdb) set $sp=0x33D50000 <- registres sp, pc et lr pointent sur le
(gdb) set $pc=0x33D50000    début du noyau en RAM
(gdb) set $lr=0x33D50000
(gdb) continue           <- exécution à partir de la prochaine
                            instruction
```

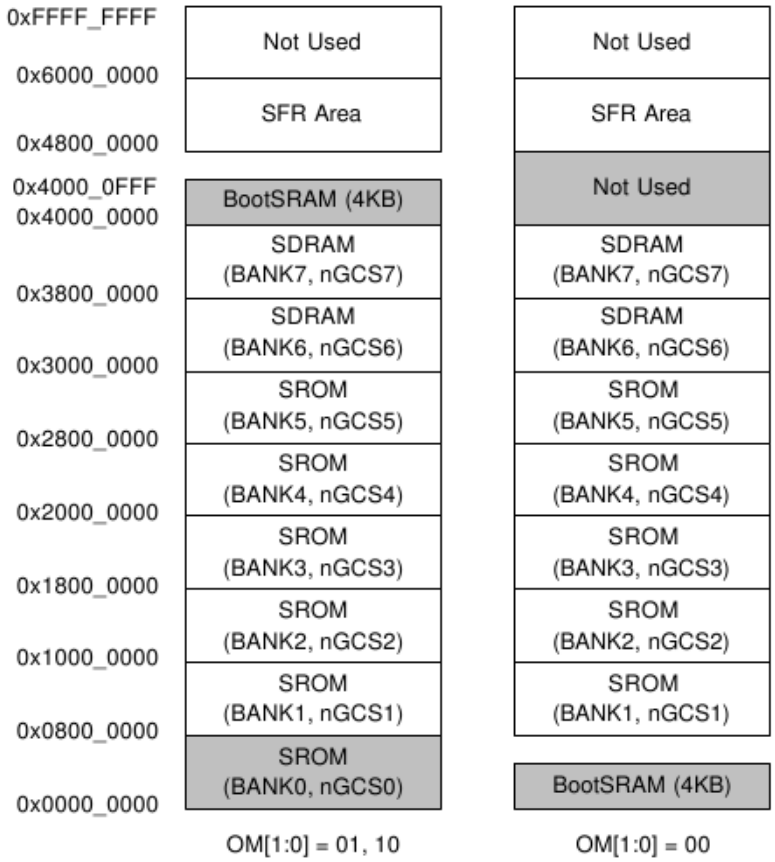
- Méthode déterminée grâce à u-boot
- L'adresse « 0x33D50000 » est déterminée après calcul (beaucoup d'autres peuvent fonctionner)
- Le kernel est chargé en RAM, se décompresse en RAM et redémarre depuis sa nouvelle adresse de décompression : il ne faut pas « s'écraser soi-même » !



# Déterminer l'adresse de chargement



- Plan de la mémoire (p221 doc CPU)
- L'image kernel de Linux se décompresse à l'adresse 0x30008000 (l'exécution de la décompression est indépendante de la position en mémoire)
- On charge en adresse haute, il faut éviter un réécrasement : 0x30008000 + 60Mo = 0x33C08000
- On arrondit un peu en supérieur, en pensant à ne pas réécraser u-boot, chargé en 0x33F80000 (ça pourrait resservir), ce qui évite aussi de dépasser 0x4000_0000 : 0x33D50000
- Attention au Steppingstone sur les 4 premiers 4Ko ! (dépend des PINs sur la carte « OM[1:0] = 00 » signifie « flash NAND active » [p215])



# Côté noyau : configuration



- .config :

```
CONFIG_ARCH_S3C2410=y
CONFIG_PLAT_S3C24XX=y
CONFIG_CPU_S3C244X=y
CONFIG_S3C2410_DMA=y
CONFIG_MACH_SMDK=y
CONFIG_PLAT_S3C=y
CONFIG_CPU_LLSERIAL_S3C2440_ONLY=y
CONFIG_CPU_LLSERIAL_S3C2440=y
CONFIG_S3C2410_PM=y
CONFIG_S3C2410_GPIO=y
CONFIG_S3C2410_CLOCK=y
CONFIG_CPU_S3C2440=y
CONFIG_S3C2440_DMA=y
CONFIG_ARCH_S3C2440=y
CONFIG_SMDK2440_CPU2440=y
CONFIG_CPU_32=y
CONFIG_CPU_ARM920T=y
CONFIG_CPU_32v4T=y
CONFIG_CPU_ABRT_EV4T=y
CONFIG_CPU_CACHE_V4WT=y
CONFIG_CPU_CACHE_VIVT=y
CONFIG_CPU_COPY_V4WB=y
CONFIG_CPU_TLB_V4WBI=y
CONFIG_CPU_CP15=y
CONFIG_CPU_CP15_MMU=y
CONFIG_MTD_NAND_S3C2410=y
CONFIG_MTD_NAND_S3C2410_DEBUG=y
CONFIG_SERIAL_S3C2410=y
CONFIG_SERIAL_S3C2410_CONSOLE=y
CONFIG_I2C_S3C2410=y
CONFIG_SPI_S3C24XX=y
CONFIG_SPI_S3C24XX_GPIO=y
CONFIG_S3C2410_WATCHDOG=y
CONFIG_FB_S3C2410=y
CONFIG_LEDS_S3C24XX=y
```

- Compatibilité a priori : S3C2410/S3C2440 et carte SMDK2440
- Dans arch/arm/mach-s3c2410/Makefile.boot:
  zreladdr-y := 0x30008000

# Côté noyau : images



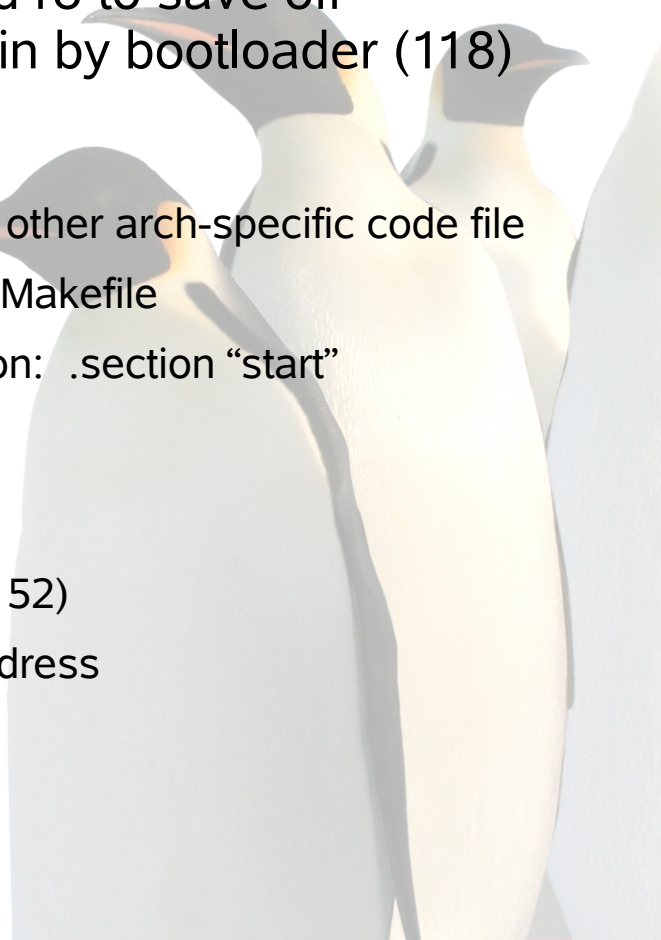
- Compilation : arch/arm/boot/compressed/vmlinux
  make ARCH=arm xconfig
  make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage
- Noyau non compressé disponible à la racine :
  31M vmlinux
  1,5M arch/arm/boot/compressed/vmlinux
- Mais alors mappé en 0xC0008000 :
  arm-none-linux-gnueabi-objdump -D ./vmlinux | less
- Le fichier System.map contient l'adresse des symboles après remapping :

```
c0004000 A swapper_pg_dir
c0008000 T __init_begin
c0008000 T _sinittext
c0008000 T stext
c0008000 T _stext
c0008034 t __enable_mmu
c0008060 t __turn_mmu_on
......
```

# Côté noyau : démarrage



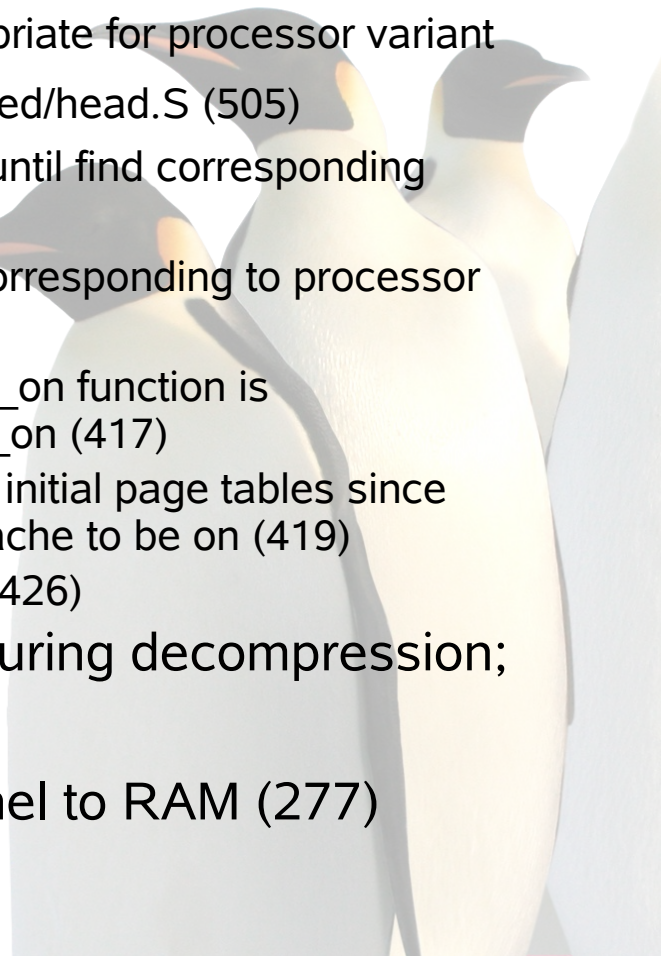
- Séquence de démarrage sur un 2.6.18
  http://gicl.cs.drexel.edu/people/sevy/linux/ARM_Linux_boot_sequence.html
- arch/arm/boot/compressed/head.S: start (108)
  - First code executed, jumped to by the bootloader, at label "start" (108)
  - save contents of registers r1 and r2 in r7 and r8 to save off architecture ID and atags pointer passed in by bootloader (118)
  - execute arch-specific code (inserted at 146)
    - arch/arm/boot/compressed/head-xscale.S or other arch-specific code file
    - added to build in arch/arm/boot/compressed/Makefile
    - linked into head.S by linker section declaration: .section “start”
    - flush cache, turn off cache and MMU
  - load registers with stored parameters (152)
    - sp = stack pointer for decompression code (152)
    - r4 = zreladdr = kernel entry point physical address



# Côté noyau : démarrage (2)



- check if running at link address, and fix up global offset table if not (196)
- zero decompression bss (205)
- call cache_on to turn on cache (218)
  - defined at arch/arm/boot/compressed/head.S (320)
  - call call_cache_fn to turn on cache as appropriate for processor variant
    - defined at arch/arm/boot/compressed/head.S (505)
    - walk through proc_types list (530) until find corresponding processor
    - call cache-on function in list item corresponding to processor (511)
      - for ARMv5tej core, cache_on function is __armv4_mmu_cache_on (417)
      - call setup_mmu to set up initial page tables since MMU must be on for cache to be on (419)
      - turn on cache and MMU (426)
- check to make sure won't overwrite image during decompression; assume not for this trace (232)
- call decompress_kernel to decompress kernel to RAM (277)



# Côté noyau : démarrage (3)



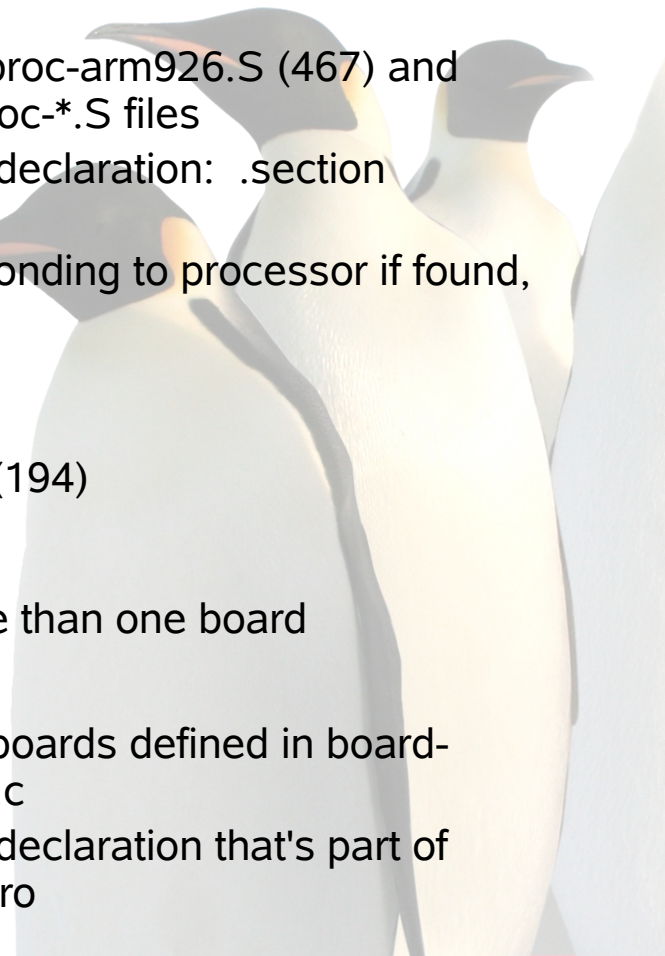
- branch to call_kernel (278)
  - call cache_clean_flush to flush cache contents to RAM (484)
  - call cache_off to turn cache off as expected by kernel initialization routines (485)
  - jump to start of kernel in RAM (489)
    - jump to address in r4 = zreladdr from previous load
      - zreladdr = ZRELADDR = zreladdr-y
      - zreladdr-y specified in arch/arm/mach-vx115/Makefile.boot

- On arrive enfin à la load address !
- Le kernel est entièrement décompressé en RAM
- Pour l'instant, aucune trace à l'écran (sauf celui indiquant la décompression) : debug au J-Tag sur les symboles obligatoire...
- Reste à exécuter le code spécifique de démarrage du CPU ARM

# Côté noyau : démarrage (4)



- arch/arm/kernel/head.S: stext (72)
  - call __lookup_processor_type (76)
    - defined in arch/arm/kernel/head-common.S (146)
    - search list of supported processor types __proc_info_begin (176)
      - kernel may be built to support more than one processor type
      - list of proc_info_list structs
        - defined in arch/arm/mm/proc-arm926.S (467) and other corresponding proc-*.S files
        - linked into list by section declaration: .section ".proc.info.init"
    - return pointer to proc_info_list struct corresponding to processor if found, or loop in error if not
  - call __lookup_machine_type (79)
    - defined in arch/arm/kernel/head-common.S (194)
    - search list of supported machines (boards)
      - kernel may be built to support more than one board
      - list of machine_desc structs
        - machine_desc struct for boards defined in board-specific file vx115_vep.c
        - linked into list by section declaration that's part of MACHINE_DESC macro

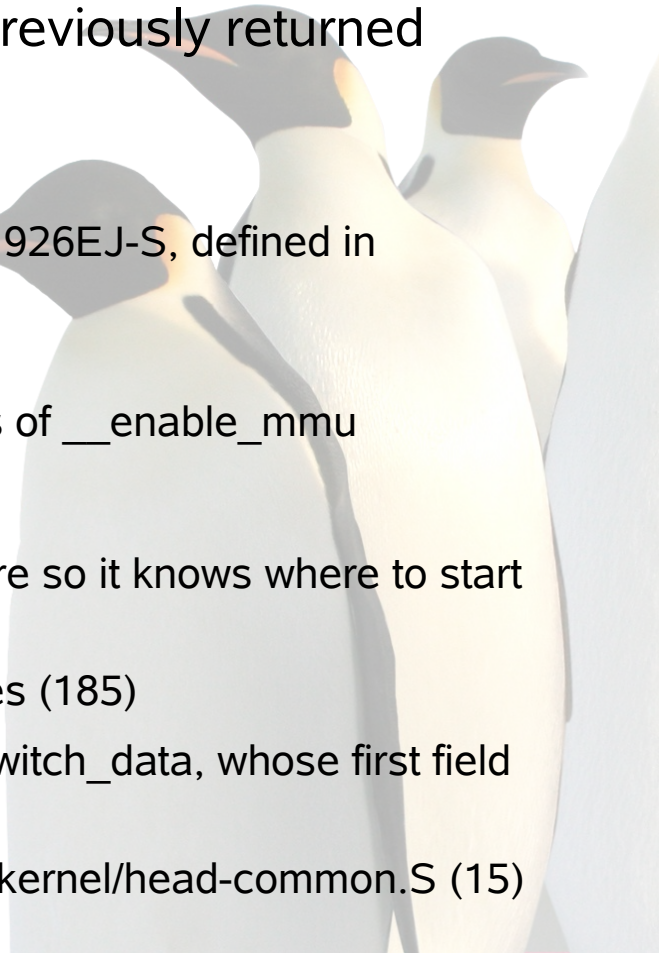


# Côté noyau : démarrage (5)



- return pointer to machine_desc struct corresponding to machine (board)

- call __create_page_tables to set up initial MMU tables (82)
- set lr to __enable_mmu, r13 to address of __switch_data (91, 93)
  - lr and r13 used for jumps after the following calls
  - __switch_data defined in arch/arm/kernel/head-common.S (15)
- call the __cpu_flush function pointer in the previously returned proc_info_list struct (94)
  - offset is #PROCINFO_INITFUNC into struct
  - this function is __arm926_setup for the ARM 926EJ-S, defined in arch/arm/mm/proc-arm926.S (392)
    - initialize caches, writebuffer
    - jump to lr, previously set to address of __enable_mmu
- __enable_mmu (147)
  - set page table pointer (TTB) in MMU hardware so it knows where to start page-table walks (167)
  - enable MMU so running with virtual addresses (185)
  - jump to r13, previously set to address of __switch_data, whose first field is address of __mmap_switched
    - __switch_data defined in arch/arm/kernel/head-common.S (15)



# Côté noyau : démarrage (6)



- arch/arm/kernel/head-common.S: __mmap_switched (35)
  - copy data segment to RAM (39)
  - zero BSS (45)
  - branch to start_kernel (55)
- La MMU a été activée (adresses à présent virtuelles)
- On démarre alors le kernel normalement, la phase spécifique est achevée : start_kernel !
- init/main.c: start_kernel (456)

# Débugage à l'initialisation

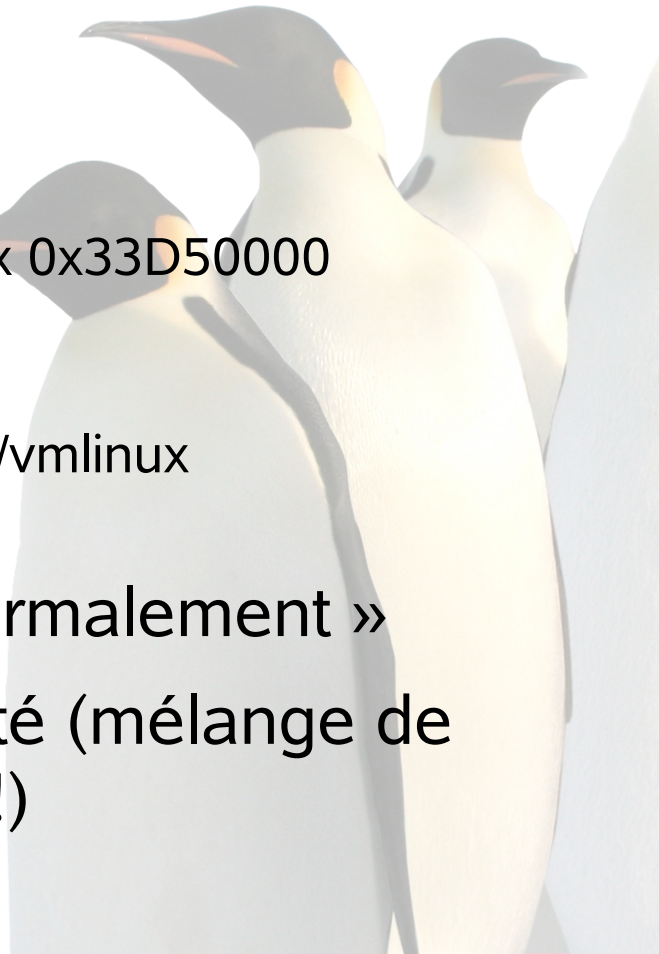
- Problème : le noyau 2.6.25 (ou 2.6.27) affiche des caractères illisibles en console
- Première idée : suivre l'initialisation du kernel
  - Avant l'activation de la MMU :

    arm-none-linux-gnueabi-gdb

    (gdb) target remote .....

    (gdb) add-symbol-file linux-2.6.25/arch/arm/boot/compressed/vmlinux 0x33D50000
  - Après l'activation de la MMU :

    $ arm-none-linux-gnueabi-gdb linux-2.6.25/vmlinux
    ou : (gdb) file linux-2.6.27.4/vmlinux
  - Possibilité alors d'utiliser gdb « normalement »
  - Attention : pas très stable en réalité (mélange de registres, voire segfault de gdb !)
    [v6.6.50 de 2007, gcc 4.2.0]

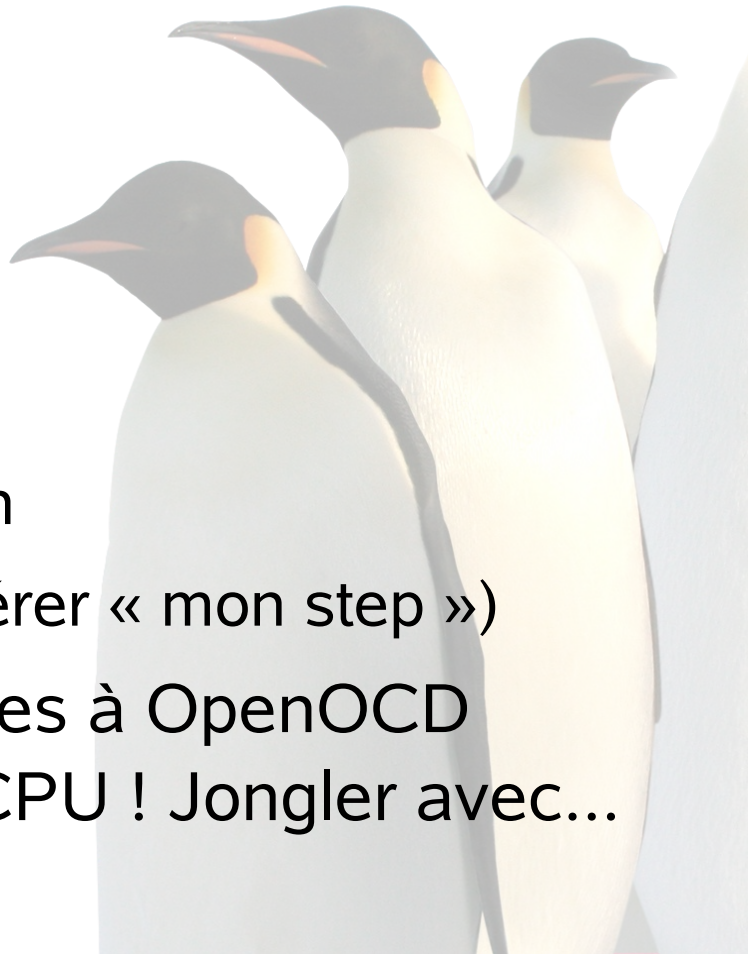


- GNU debugger, projet GCC/binutils
- http://www.gnu.org/software/gdb/
- Pour l'embarqué : gdbserver
  - http://davis.lbl.gov/Manuals/GDB/gdb_17.html
  - Quelques ko sur la cible
  - Communique avec gdb distant
- Crosscompilé :
  - debug à distance (via gdb-server) d'un binaire pour une archi cible, en restant sur la machine de développement (avec toutes les fonctionnalités)
  - Buildroot pour la compilation
  - Codesourcery en précompilé (étrangement, marche mieux)

# Utilisation de gdb via le J-Tag



- Commandes de gdb :
  - break (b) : break point hardware (et « watch »)
  - print/printf : affichage
  - disassemble : affichage d'une région de la mémoire désassemblée
  - dump : dump de la mémoire
  - *etc*
- Intéraction avec gdb :
  - Ctrl-C pour arrêter l'exécution
  - continue (c) : continuer l'exécution
  - step (s)/next (n) : pas à pas (préférer « mon step »)
- Gdb transmet en réalité les commandes à OpenOCD
- Seulement deux break points sur ce CPU ! Jongler avec...

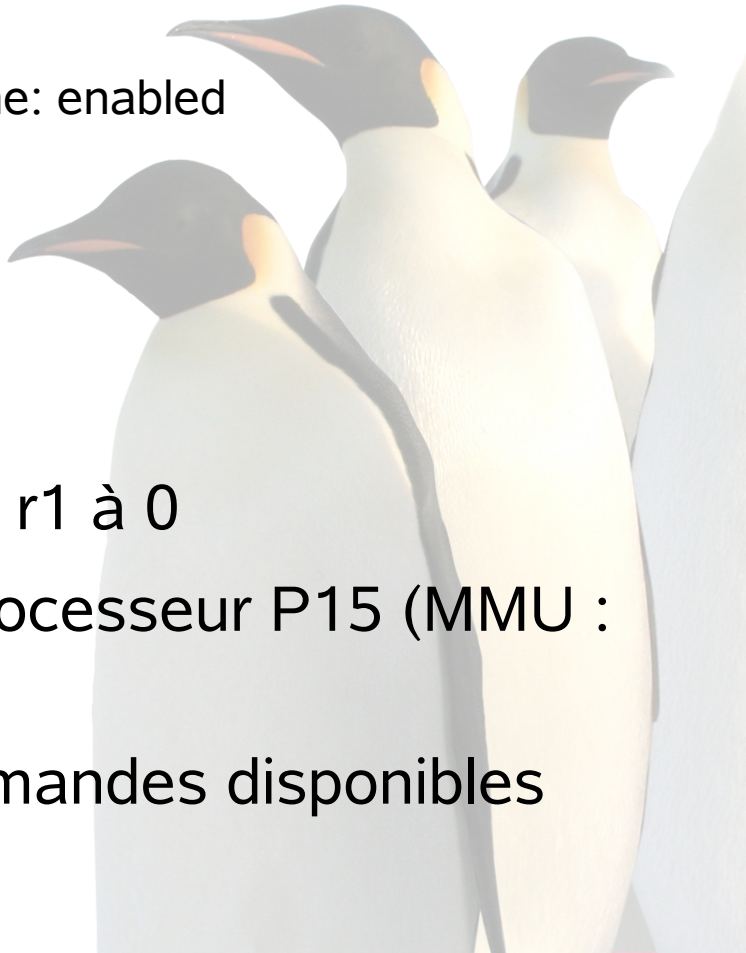


# Côté OpenOCD

- Traduction des commandes gdb en actions hardware
- Possibilité de passer des commandes directes depuis gdb :
  - mon poll

    target state: halted
    target halted in ARM state due to debug request, current mode: Supervisor
    cpsr: 0x40000053 pc: 0x33d50000
    MMU: disabled, D-Cache: enabled, I-Cache: enabled
  - mon reg

    (0) r0 (/32): 0x00000000 (dirty: 1, valid: 1)
    (1) r1 (/32): 0x0000016a (dirty: 1, valid: 1)
    (2) r2 (/32): 0x00000000 (dirty: 1, valid: 1)
    //etc//
  - mon reg r1 0x0 : passe le registre r1 à 0
  - mon arm920t cp15 : gérer le coprocesseur P15 (MMU : infructueux à l'usage)
  - mon help : affiche toutes les commandes disponibles



# Debug sans la console



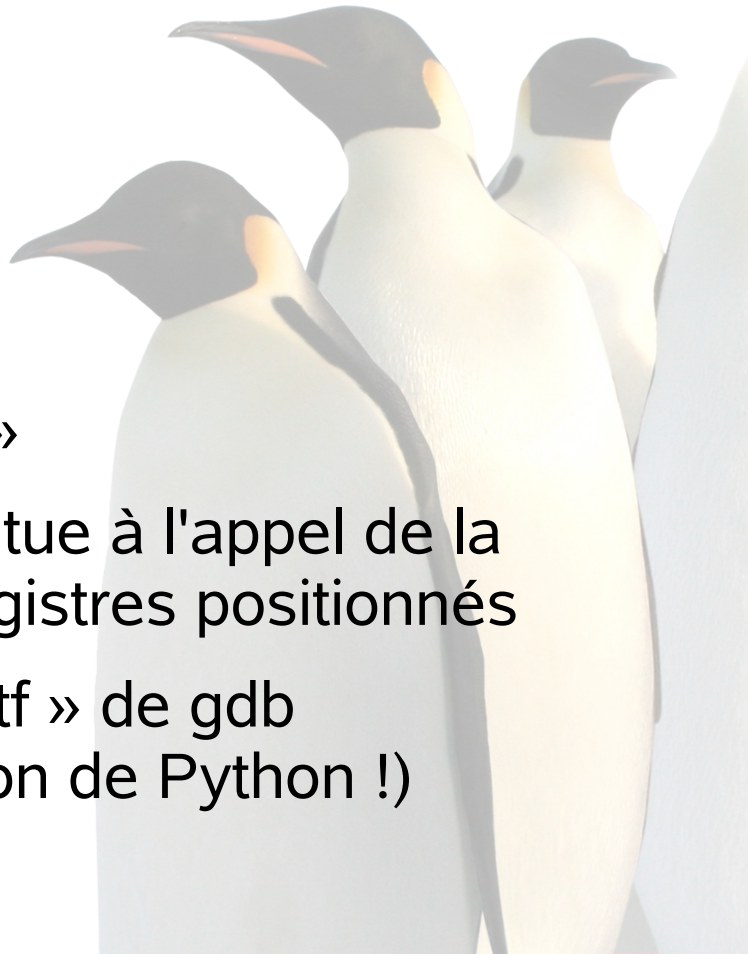
- Idée : breakpoint sur « printk », récupération de l'affichage avec gdb
- Mise en oeuvre :
  - Récupération de l'adresse mémoire de printk :

    ```
    > grep " printk$" System.map
    c003deac T printk
    ```

  - Breakpoint sur printk :

    ```
    (gdb) b *0xc003deac
    Breakpoint 1 at 0xc003deac
    (gdb)
    ```

  - Lancer le kernel avec « continue »
  - À chaque arrêt du kernel, on se situe à l'appel de la fonction printk, avec tous les registres positionnés
  - Utilisation de la commande « printf » de gdb (fastidieux : vivement l'intégration de Python !)



# Exemple en console gdb



```
(gdb) c
Continuing.

Breakpoint 1, 0xc003deac in ?? ()
(gdb) printf "%s",$r0
Linux version 2.6.27.4linagora (gblanc@deepblue) (gcc version 4.2.0 20070413 (prerelease)
(CodeSourcery Sourcery G++ Lite 2007q1-10)) #4 PREEMPT Wed Oct 29 19:33:38 CET 2008
(gdb) c
Continuing.

Breakpoint 1, 0xc003deac in ?? ()
(gdb) printf "%s",$r0
CPU: %s [%08x] revision %d (ARMv%s), cr=%08lx
(gdb) c
Continuing.

Breakpoint 1, 0xc003deac in ?? ()
(gdb) printf "%s",$r0
Machine: %s
(gdb) printf "%s",$r1
SMDK2440(gdb) c
Continuing.
```

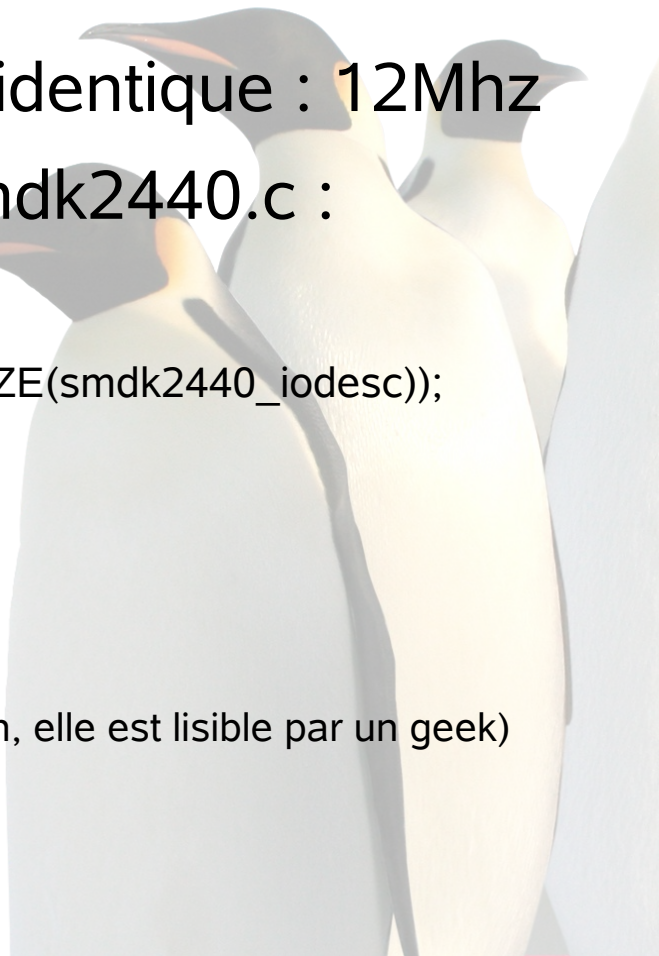
# 2nd problème : Problématique de migration



- L'initialisation se passe donc correctement, mais le port série affiche n'importe quoi (caractères illisibles)
- Seconde idée : comparer u-boot (qui fonctionne correctement), le noyau 2.6.13 patché par le fournisseur et le noyau Linux vanilla :
  - On trouve que l'horloge n'est pas identique : 12Mhz
  - arch/arm/mach-s3c2440/mach-smdk2440.c :

```
static void __init smdk2440_map_io(void)
    {
        s3c24xx_init_io(smdk2440_iodesc, ARRAY_SIZE(smdk2440_iodesc));
    //  s3c24xx_init_clocks(16934400);
        s3c24xx_init_clocks(12000000);
        s3c24xx_init_uarts(smdk2440_uartcfgs,
    ARRAY_SIZE(smdk2440_uartcfgs));
    }
```

- La console parle alors en humain ! (enfin, elle est lisible par un geek)



# Mise en place de l'environnement de dev

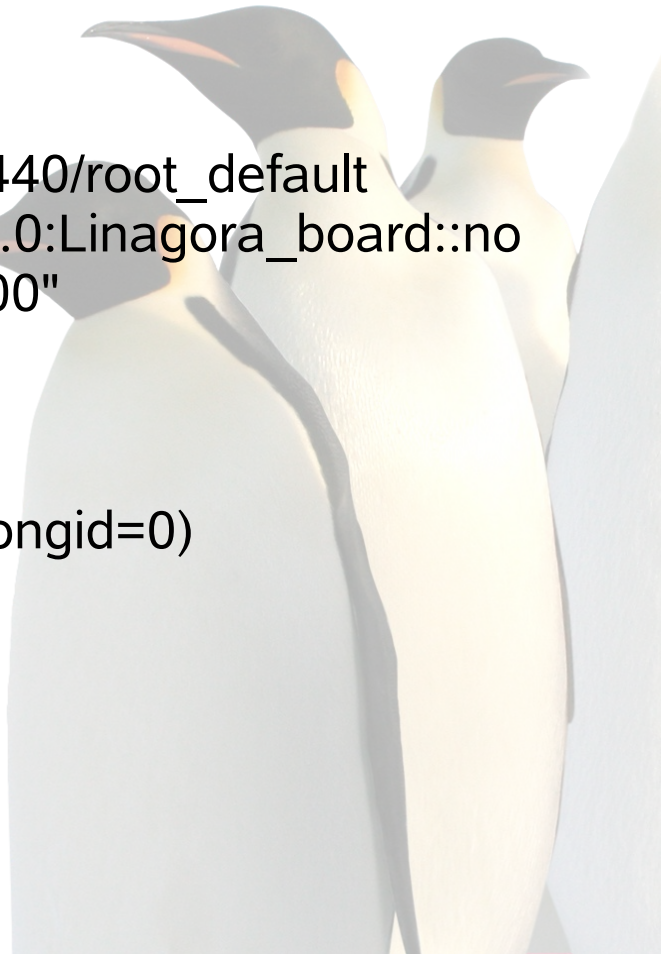


- Boot sur réseau :
  - Récupération d'un système avec EABI (Ångström, buildroot, peu importe)
  - Configuration du NFS :
    - Côté kernel cible :

```
CONFIG_CMDLINE="root=/dev/nfs
    nfsroot=192.168.1.1:/home/gblanc/S3C2440/root_default
    ip=192.168.1.2:192.168.1.1::255.255.255.0:Linagora_board::no
    ne ro init=/linuxrc console=ttySAC0,115200"
```

    - Côté « exports » :

```
/home/gblanc/S3C2440/root_default
    (rw,no_root_squash,async,anonuid=0,anongid=0)
```

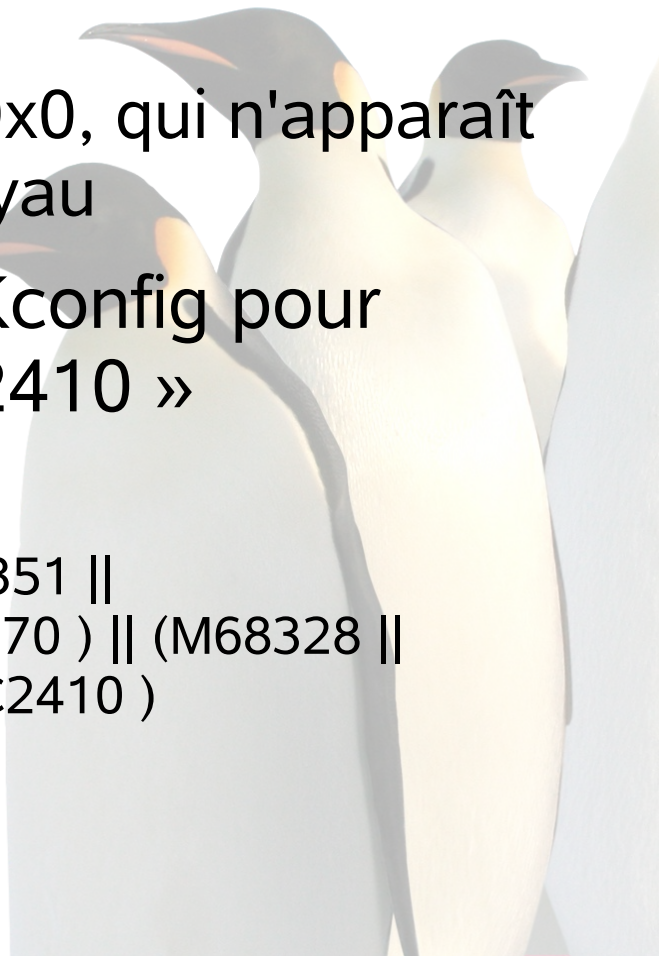


# Problématique de migration : encore



- Le réseau ne fonctionne pas !
  - Sur le kernel 2.6.13 fourni, le driver CS8900 (Cirrus) marche (certainement patché)
  - Mais il n'existe plus, et n'est pas transposable sans y passer beaucoup de temps
  - Il existe un driver générique CS89x0, qui n'apparaît pas dans la configuration du noyau
- Première étape : patcher drivers/net/Kconfig pour activer CS89x0 avec « || ARCH_S3C2410 »

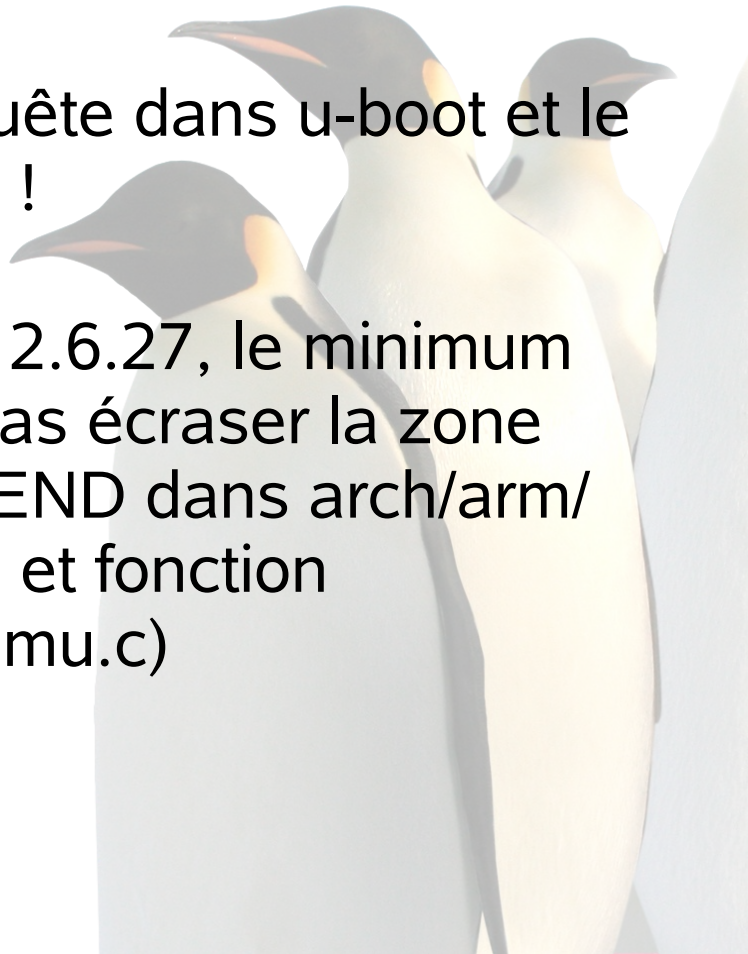
```
config CS89x0
        tristate "CS89x0 support"
        depends on NET_PCI && (ISA || MACH_IXDP2351 ||
ARCH_IXDP2X01 || ARCH_PNX010X || MACH_DM270 ) || (M68328 ||
M68EZ328 || M68VZ328 || COLDFIRE  || ARCH_S3C2410 )
```



# Mapper le réseau



- Mapper (en MMU) les adresses physiques du réseau en mémoire virtuelle
- C'est-à-dire : 0x18000000 + 0x01000000 (soit 0x19000000) sur 0xE0000000, avec une taille « SZ_1M » de 1Mo
- L'adresse de base se trouve après enquête dans u-boot et le noyau 2.6.13 fourni : la doc n'en dit rien !
- 0xE0000000 est a priori, dans le noyau 2.6.27, le minimum en adresses virtuelles hautes pour ne pas écraser la zone virtuelle de malloc (macro VMALLOC_END dans arch/arm/mach-s3c2410/include/mach/vmalloc.h, et fonction « create_mapping » de arch/arm/mm/mmu.c)



# Mapper le réseau (2)

- 0xE0000000 marche fort bien : dans arch/arm/mach-s3c2440/mach-smdk2440.c

```
static struct map_desc smdk2440_iodesc[] __initdata = {
/* ISA IO Space map (memory space selected by A24) */
{
        .virtual        = (u32)S3C24XX_VA_ISA_WORD,
        .pfn            = __phys_to_pfn(S3C2410_CS2),
        .length         = 0x10000,
        .type           = MT_DEVICE,
}, {
        .virtual        = (u32)S3C24XX_VA_ISA_WORD + 0x10000,
        .pfn            = __phys_to_pfn(S3C2410_CS2 + (1<<24)),
        .length         = SZ_4M,
        .type           = MT_DEVICE,
}, {
        .virtual        = (u32)S3C24XX_VA_ISA_BYTE,
        .pfn            = __phys_to_pfn(S3C2410_CS2),
        .length         = 0x10000,
        .type           = MT_DEVICE,
}, {
        .virtual        = (u32)S3C24XX_VA_ISA_BYTE + 0x10000,
        .pfn            = __phys_to_pfn(S3C2410_CS2 + (1<<24)),
        .length         = SZ_4M,
        .type           = MT_DEVICE,
},
{ 0xE0000000, __phys_to_pfn(S3C2410_CS3+0x01000000), SZ_1M, MT_DEVICE }
};
```

# Enregistrer le driver réseau



- Dans le même fichier de définition de l'architecture, il faut préciser l'activation de ce device

```
static struct platform_device *smdk2440_devices[] __initdata = {
&s3c_device_usb,
&s3c_device_lcd,
&s3c_device_wdt,
&s3c_device_i2c,
&s3c_device_iis,
&s3c_device_cs89x0,
};
```

- Il faut donc créer cette structure de déclaration du device, et la déclarer dans include/asm/plat-s3c24xx/devs.h (rajouter « extern struct platform_device s3c_device_cs89x0; »)

# Enregistrer le driver réseau (2)



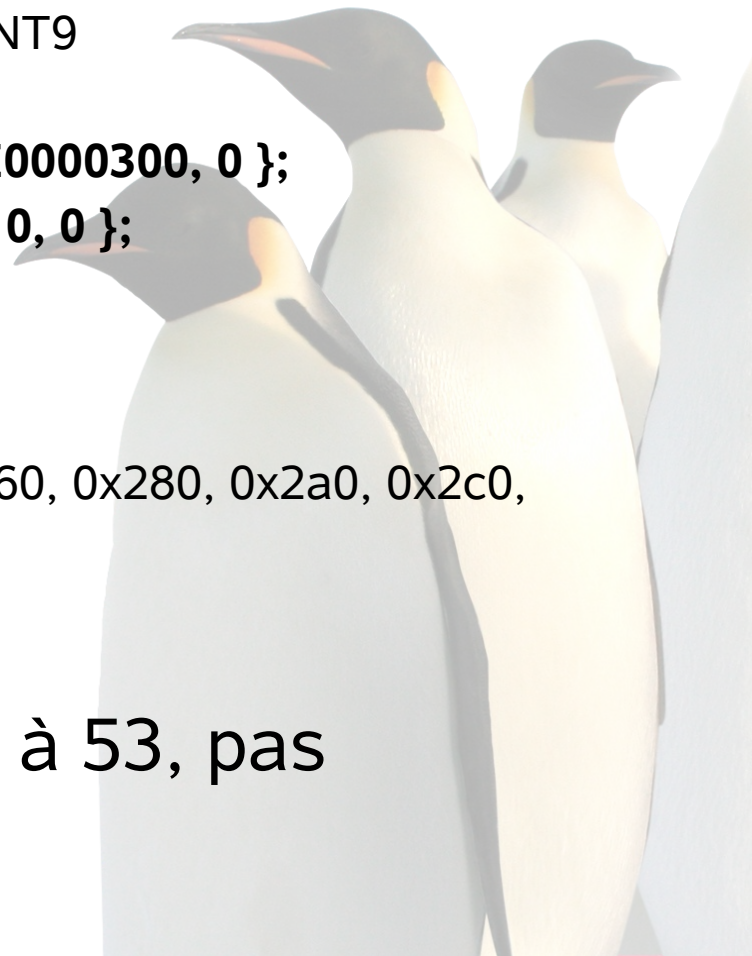
- Dans arch/arm/plat-s3c24xx/devs.c (de préférence avec du code spécifique à CONFIG_CPU_S3C2440) :

```
/* CS8900 */

static struct resource s3c_cs89x0_resource[] = {
	[0] = {
		.start  = 0x19000000,
		.end    = 0x19000000 + 16,
		.flags  = IORESOURCE_MEM,
	},
	[1] = {
		.start  = IRQ_EINT9,
		.end    = IRQ_EINT9,
		.flags  = IORESOURCE_IRQ,
	},
};

struct platform_device s3c_device_cs89x0 = {
	.name           = "cirrus-cs89x0",
	.num_resources  = ARRAY_SIZE(s3c_cs89x0_resource),
	.resource       = s3c_cs89x0_resource,
};

EXPORT_SYMBOL(s3c_device_cs89x0);
```

- Dans le driver CS89x0 (drivers/net/cs89x0.c) :

```
#elif defined(CONFIG_ARCH_PNX010X)
// blablabla
#elif defined(CONFIG_ARCH_S3C2410)
#include <asm/arch/irqs.h>  // copie de asm-arm/arch-s3c2410/irqs.h dans 2.6.25 ; attention, dans 2.6.27 se sera <mach/irqs.h> qui pointe vers arch/arm/mach-s3c2410/include/mach/irqs.h ; il faut trouver où est IRQ_EINT9
#include <linux/irq.h>
static unsigned int netcard_portlist [] __initdata = { 0xE0000300, 0 };
static unsigned int cs8900_irq_map[] = { IRQ_EINT9, 0, 0, 0 };
#define NO_EPROM
#else
static unsigned int netcard_portlist[] __used __initdata =
   { 0x300, 0x320, 0x340, 0x360, 0x200, 0x220, 0x240, 0x260, 0x280, 0x2a0, 0x2c0, 0x2e0, 0};
static unsigned int cs8900_irq_map[] = {10,11,12,5};
#endif
```

- Base mémoire à 0xE0000300, IRQ à 53, pas d'EPROM



# Brancher le réseau (2)

- Sur la 2.6.27, la macro NO_EPROM a disparu : la remettre sous peine de Oops !
- Pas d'EPROM, donc nécessité de renseigner l'adresse mac à la main, dans la fonction cs89x0_probe1 (trouver un système de macro si plusieurs cartes sont en jeu ! Ou renseigner via la ligne de commande de démarrage du noyau, mais une valeur par défaut évite le Oops) :

```
printk(" IRQ %d", dev->irq);

dev->dev_addr[0] = 0x00;
dev->dev_addr[1] = 0x00;
dev->dev_addr[2] = 0xc0;
dev->dev_addr[3] = 0xff;
dev->dev_addr[4] = 0xee;
dev->dev_addr[5] = 0x08;
set_mac_address(dev, dev->dev_addr);
```

# Brancher le réseau (3)

- Patcher pour renseigner l'IRQ :
  - Dans le probe :

```
 if (lp->chip_type == CS8900) {
#if defined(CONFIG_MACH_IXDP2351) || defined(CONFIG_ARCH_IXDP2X01) ||
defined(CONFIG_ARCH_PNX010X) || defined(CONFIG_ARCH_S3C2410)
                i = cs8900_irq_map[0];
#else
```

  - Dans net_open() :

```
#if !defined(CONFIG_MACH_IXDP2351) && !defined(CONFIG_ARCH_IXDP2X01) && !defined(CONFIG_ARCH_PNX010X) && !
defined(CONFIG_ARCH_S3C2410)
        if (((1 << dev->irq) & lp->irq_map) == 0) {
            printk(KERN_ERR "%s: IRQ %d is not in our map of allowable IRQs, which is %x\n",
                dev->name, dev->irq, lp->irq_map);
            ret = -EAGAIN;
            goto bad_out;
        }
#endif

        set_irq_type(dev->irq, IRQ_TYPE_EDGE_RISING);   // IRQT_RISING dans les vieux kernels
/* FIXME: Cirrus' release had this: */
        writereg(dev, PP_BusCTL, readreg(dev, PP_BusCTL)|ENABLE_IRQ );
/* And 2.3.47 had this: */
#if 0
        writereg(dev, PP_BusCTL, ENABLE_IRQ | MEMORY_ON);
#endif
        write_irq(dev, lp->chip_type, dev->irq);
        ret = request_irq(dev->irq, &net_interrupt, 0, dev->name, dev);
```

# Brancher le réseau (4)

- Ça marche enfin !

eth0: 10Base-T (RJ-45) has no cable
eth0: using half-duplex 10Base-T (RJ-45)
IP-Config: Complete:
device=eth0, addr=192.168.1.2, mask=255.255.255.0, gw=255.255.255.255,
host=Linagora_board, domain=, nis-domain=(none),
bootserver=192.168.1.1, rootserver=192.168.1.1, rootpath=
Looking up port of RPC 100003/2 on 192.168.1.1

- Il n'y a plus les erreurs

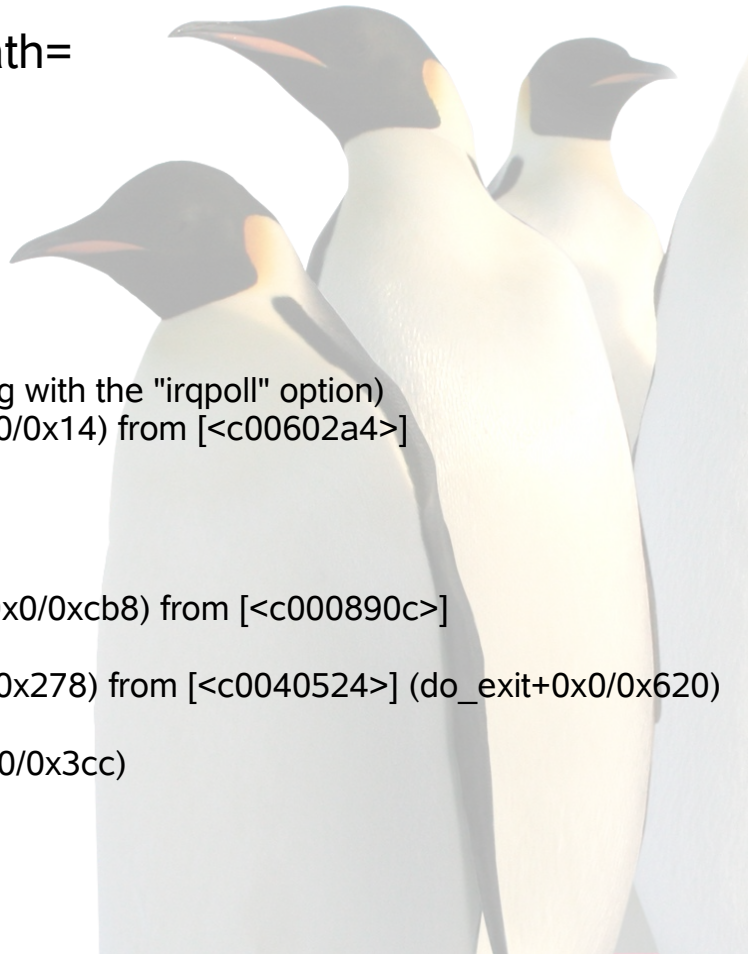
Looking up port of RPC 100003/2 on 192.168.1.1
eth0: sent 42 byte packet of type 806
NETDEV WATCHDOG: eth0: transmit timed out
eth0: transmit timed out, IRQ conflict ??
eth0: sent 42 byte packet of type 806
cs89x0: Tx buffer not free!
NETDEV WATCHDOG: eth0: transmit timed out
eth0: transmit timed out, IRQ conflict ??
eth0: sent 42 byte packet of type 806

## Ou encore :

irq 53: nobody cared (try booting with the "irqpoll" option)
[<c0027d68>] (dump_stack+0x0/0x14) from [<c00602a4>]
(__report_bad_irq+0x38/0x88)

// etc //

[<c001cf3c>] (ip_auto_config+0x0/0xcb8) from [<c000890c>]
(kernel_init+0xb8/0x278)
[<c0008854>] (kernel_init+0x0/0x278) from [<c0040524>] (do_exit+0x0/0x620)
handlers:
[<c016811c>] (net_interrupt+0x0/0x3cc)
Disabling IRQ #53



# Le miracle (to be continued)



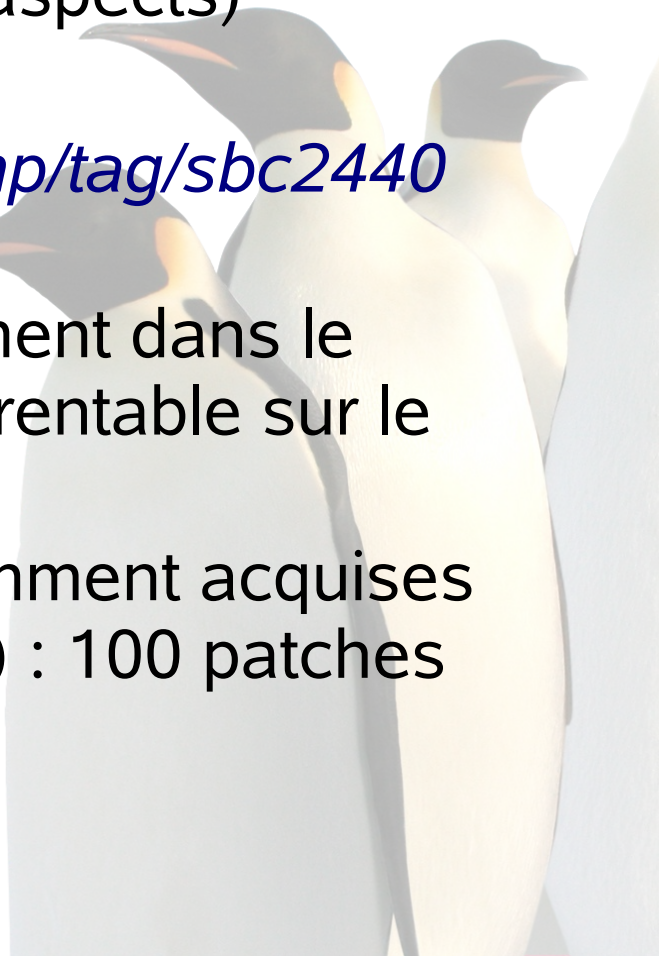
- On boote « correctement », avec de vrais logs
- On charge à la fin le programme indiqué par « init= » en ligne de commande
- On boote depuis la RAM ! (mais transfert J-Tag long, avec du bas de gamme)
- Mais le mapping de la flash est en dur dans le code de l'architecture : smdk_default_nand_part, très indirectement appelé par la fonction smdk_machine_init, dans arch/arm/plat-s3c24xx/common-smdk.c  (le jeu de piste continue...)

# Conclusion



- Une semaine et demi à raison de 2 ou 3 heures par jour
- Un passage de 2.6.25 à 2.6.27 pas si évident
- http://lxr.linux.no/ est l'invention du siècle
- Linux est un jeu de piste, u-boot est beaucoup plus simple (mais moins portable/modulaire sur ces aspects)
- Penser à échanger/communiquer : *http://gblanc.blogs.linagora.com/index.php/tag/sbc2440*

- Pas eu le temps de pousser un reversement dans le noyau (prend beaucoup de temps, mais rentable sur le long terme)
- Deux nouvelles cartes (Armadeus) récemment acquises sur processeur ARM9 (i.Mx27 Freescale) : 100 patches kernel pour le support de l'architecture !



# Merci de votre attention

**Contact :**
LINAGORA – Siège social
27 rue de Berri
75008 Paris – France
Tél. : +33 1 58 18 68 28
Fax : +33 1 58 18 68 29
Mail : info@linagora.com
Web : www.linagora.com