ENSEIRB-MATMECA

MISE EN ŒUVRE DE LINUX EMBARQUE TEMPS REEL SUR PROCESSEUR ARM

Patrice KADIONIK
http://kadionik.vvv.enseirb-matmeca.fr/

TABLE DES MATIERES

1 BUT DES TRAVAUX PRATIQUES

Le but de ces Travaux Pratiques est d'étudier la mise en œuvre de Linux embarqué Temps Réel avec Xenomai sur une carte d'évaluation Eukréa ARM AT91RM9200. Ces TP ont été créés en 2015.

La mise en œuvre de Linux embarqué Temps Réel sur la carte AT91RM9200 ou plus simplement AT91 a fait l'objet d'un sujet de « projets avancés » de l'option Systèmes Embarqués SE. Je tiens ainsi à remercier Pierrick Guitter, François Turban, Guillaume Vermeulen et Alexandre Sorita de la promotion SE 2014-2015 pour leur travail et leur contribution à l'amélioration constante de l'enseignement de l'option SE…

On verra les points suivants :

- Mise en œuvre de Linux embarqué.
- Mise en œuvre de l'extension Temps Réel dur Xenomai pour Linux embarqué.
- Mesures de temps de latence.

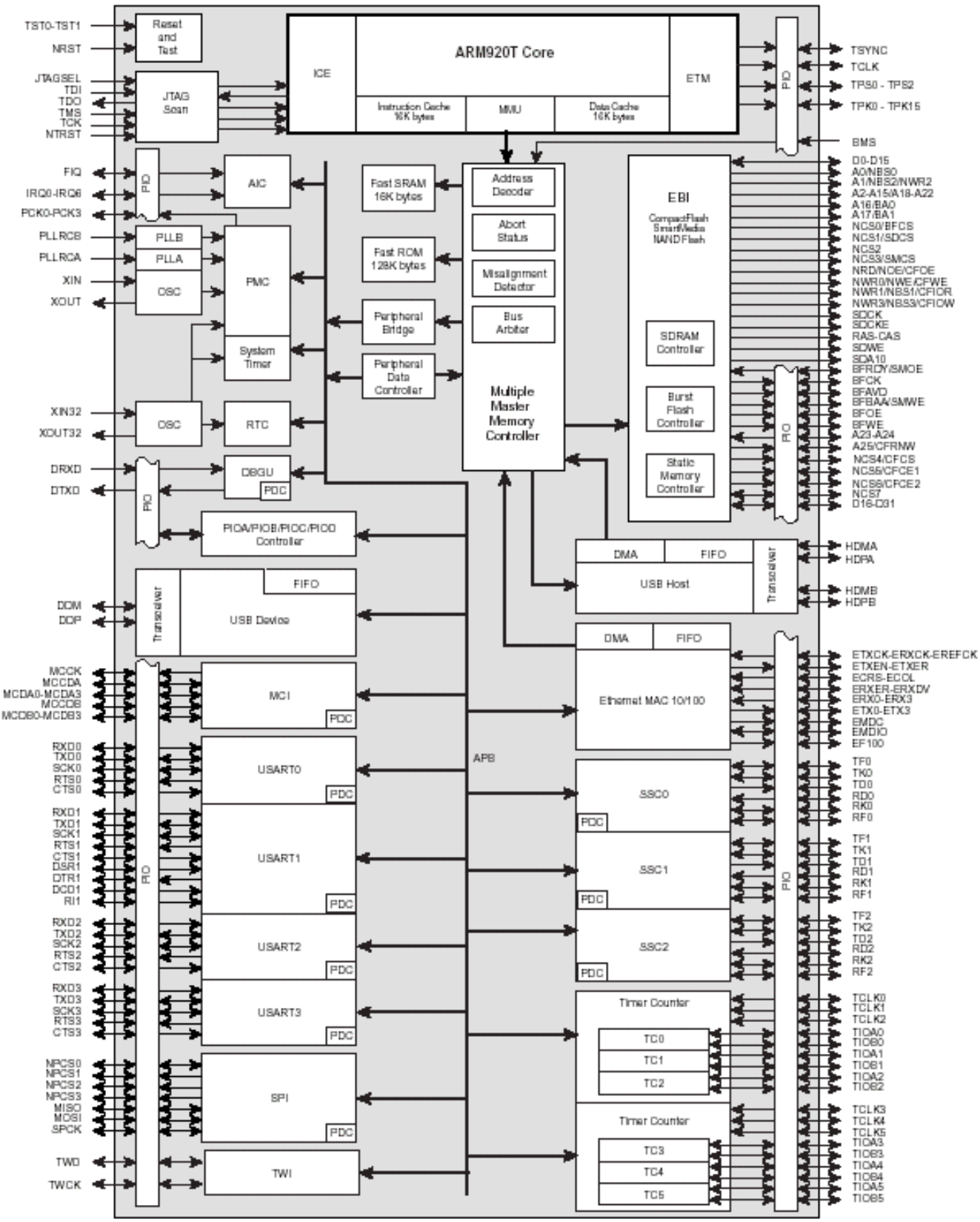
A tout moment, on se référera à l'aide contenue en annexe dans ce manuel ou bien à l'aide en ligne :

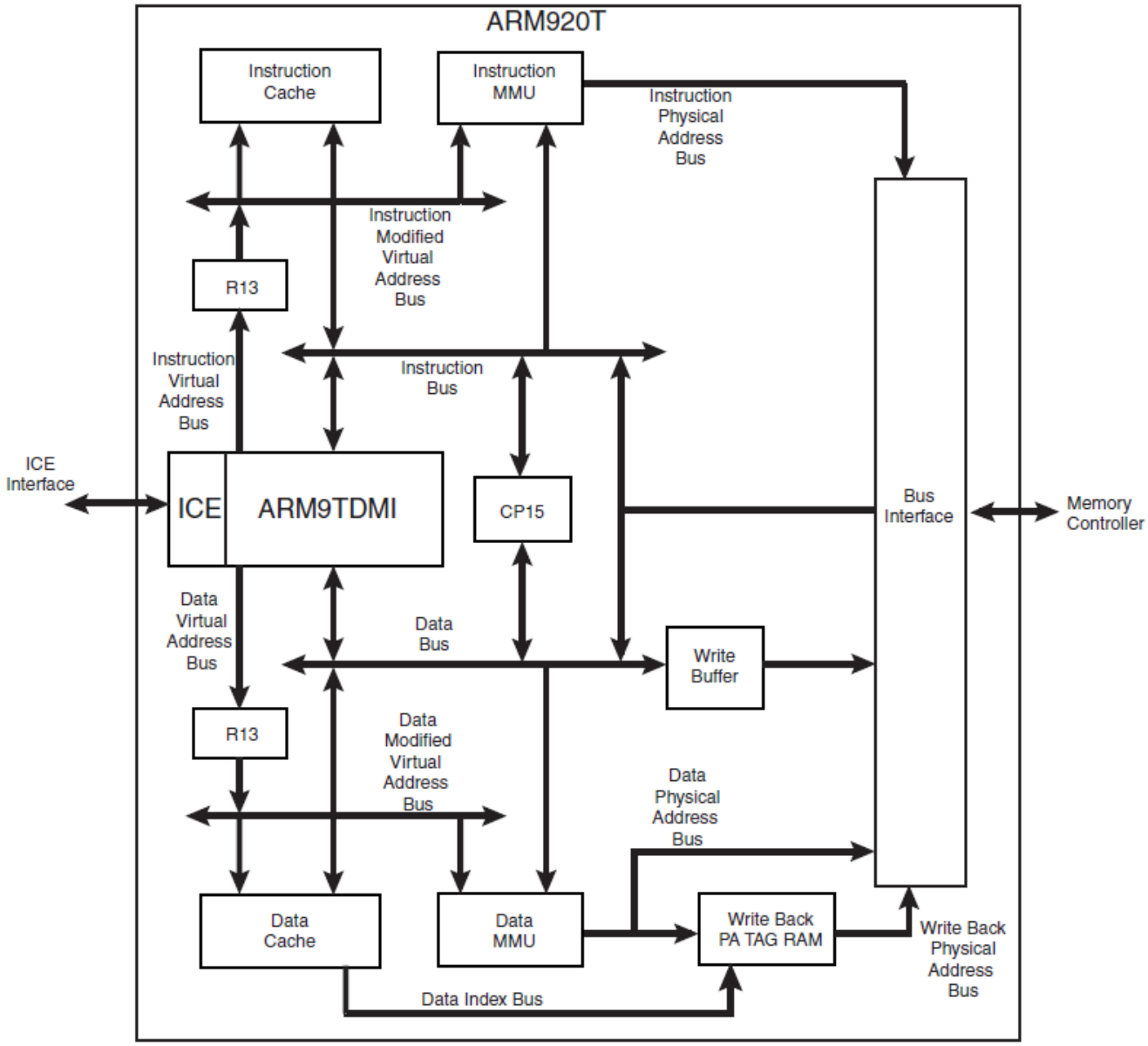
```
% man …
```

2 INFORMATIONS ESSENTIELLES SUR LA CARTE ARM AT91

La carte cible AT91 est une carte de développement de la société bordelaise Eukréa permettant de mettre en œuvre le processeur ARM9 AT91RM9200. Le processeur AT91RM9200 supporte Linux avec MMU que nous allons utiliser durant ces TP.

L'architecture du processeur AT91RM9200 est donnée sur les figures suivantes :


TST0-TST1
NRST
JTAGSEL
TDI
TDO
TMS
TCK
NTRST
Reset and Test
JTAG Scan
ICE
ARM920T Core
Instruction Cache 16K bytes
MMU
Data Cache 16K bytes
ETM
PID
TSYNC
TCLK
TPS0 - TPS2
TPK0 - TPK15
BMS
FIQ
IRQ0-IRQ6
PCK0-PCK3
AIC
Fast SRAM 16K bytes
Address Decoder
Abort Status
Misalignment Detector
Bus Arbiter
Multiple Master Memory Controller
EBI
CompactFlash SmartMedia NANDFlash
SDRAM Controller
Burst Flash Controller
Static Memory Controller
D0-D15
A0/NBS0
A1/NBS2/NWR2
A2-A15/A18-A22
A16/BA0
A17/BA1
NCS0/BFCS
NCS1/SDCS
NCS2
NCS3/SMCS
NRD/NOE/CFOE
NWR0/NWE/CFWE
NWR1/NBS1/CFIOR
NWR3/NBS3/CFIOW
SDCK
SDCKE
RAS-CAS
SDWE
SDA10
BFRDY/SMOE
BFCK
BFAVD
BFBAA/SMWE
BFOE
BFWE
A23-A24
A25/CFRNW
NCS4/CFCS
NCS5/CFCE1
NCS6/CFCE2
NCS7
D16-D31
PLLRCB
PLLRCA
XIN
XOUT
PLLB
PLLA
OSC
PMC
System Timer
Fast ROM 128K bytes
Peripheral Bridge
Peripheral Data Controller
XIN32
XOUT32
OSC
RTC
DRXD
DTXD
DBGU
PDC
PIOA/PIOB/PIOC/PIOD Controller
DMA
FIFO
USB Host
Transceiver
HDMA
HDPA
HDMB
HDPB
DDM
DDP
USB Device
MCCK
MCCDA
MCDA0-MCDA3
MCCDB
MCDB0-MCDB3
MCI
Ethernet MAC 10/100
ETXCK-ERXCK-EREFCK
ETXEN-ETXER
ECRS-ECOL
ERXER-ERXDV
ERX0-ERX3
ETX0-ETX3
EMDC
EMDIO
EF100
RXD0
TXD0
SCK0
RTS0
CTS0
USART0
APB
SSC0
TF0
TK0
TD0
RD0
RK0
RF0
RXD1
TXD1
SCK1
RTS1
CTS1
DSR1
DTR1
DCD1
RI1
USART1
SSC1
TF1
TK1
TD1
RD1
RK1
RF1
RXD2
TXD2
SCK2
RTS2
CTS2
USART2
SSC2
TF2
TK2
TD2
RD2
RK2
RF2
RXD3
TXD3
SCK3
RTS3
CTS3
USART3
Timer Counter
TC0
TC1
TC2
TCLK0
TCLK1
TCLK2
TIOA0
TIOB0
TIOA1
TIOB1
TIOA2
TIOB2
NPCS0
NPCS1
NPCS2
NPCS3
MISO
MOSI
SPCK
SPI
Timer Counter
TC3
TC4
TC5
TCLK3
TCLK4
TCLK5
TIOA3
TIOB3
TIOA4
TIOB4
TIOA5
TIOB5
TWD
TWCK
TWI



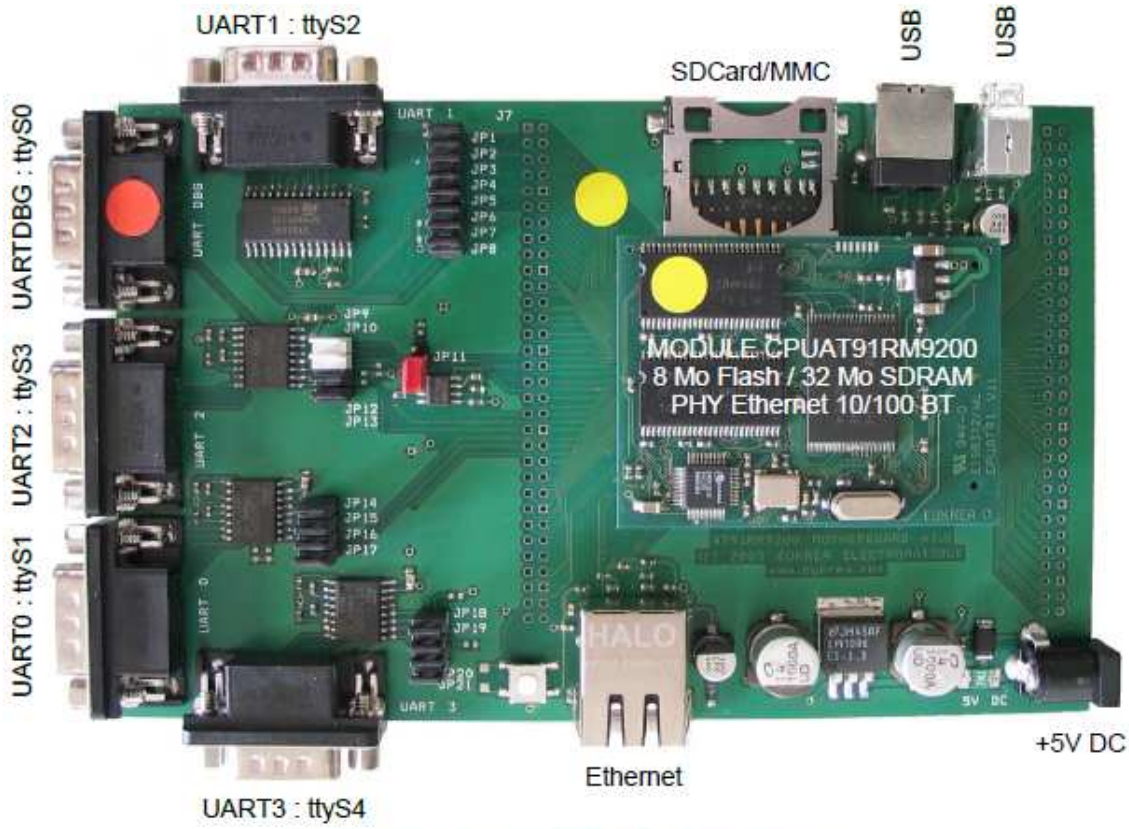
Architecture du processeur AT91RM9200

La carte cible possède les fonctionnalités suivantes :

- ARM920T ARM Thumb Processor, v4T Architecture
  - 200 MIPS at 180 MHz
  - Memory Management Unit
  - 16-KByte Data Cache, 16-KByte Instruction Cache
  - In-circuit Emulator including Debug Communication Channel
  - 16K Bytes of SRAM and 128K Bytes of ROM
  - Ethernet MAC 10/100 Base-T
  - USB 2.0 Full Speed
  - Multimedia Card Interface
  - 4 Universal Synchronous/Asynchronous Receiver/Transmitters
  - Master/Slave Serial Peripheral Interface SPI
  - Two 3-channel, 16-bit Timer/Counters
  - IEEE 1149.1 JTAG Boundary Scan
- Synchronous dynamic random access memory (SDRAM)
  - 32MB (MICRON - 128Mb x 2)
- Flash memory
  - 8 MB (MICRON MT28F640J3)
- Ethernet interface
  - 10-BaseT (10 Mb/s) and 100-BaseT (100 Mb/s) Ethernet Media Access Controller (MAC)
  - MICREL KS8721BL
- 5 Universal asynchronous receiver/transmitter (UART)
- USB host and USB device connectors
- SD/MMC slot
- 1 Push button : 1 reset

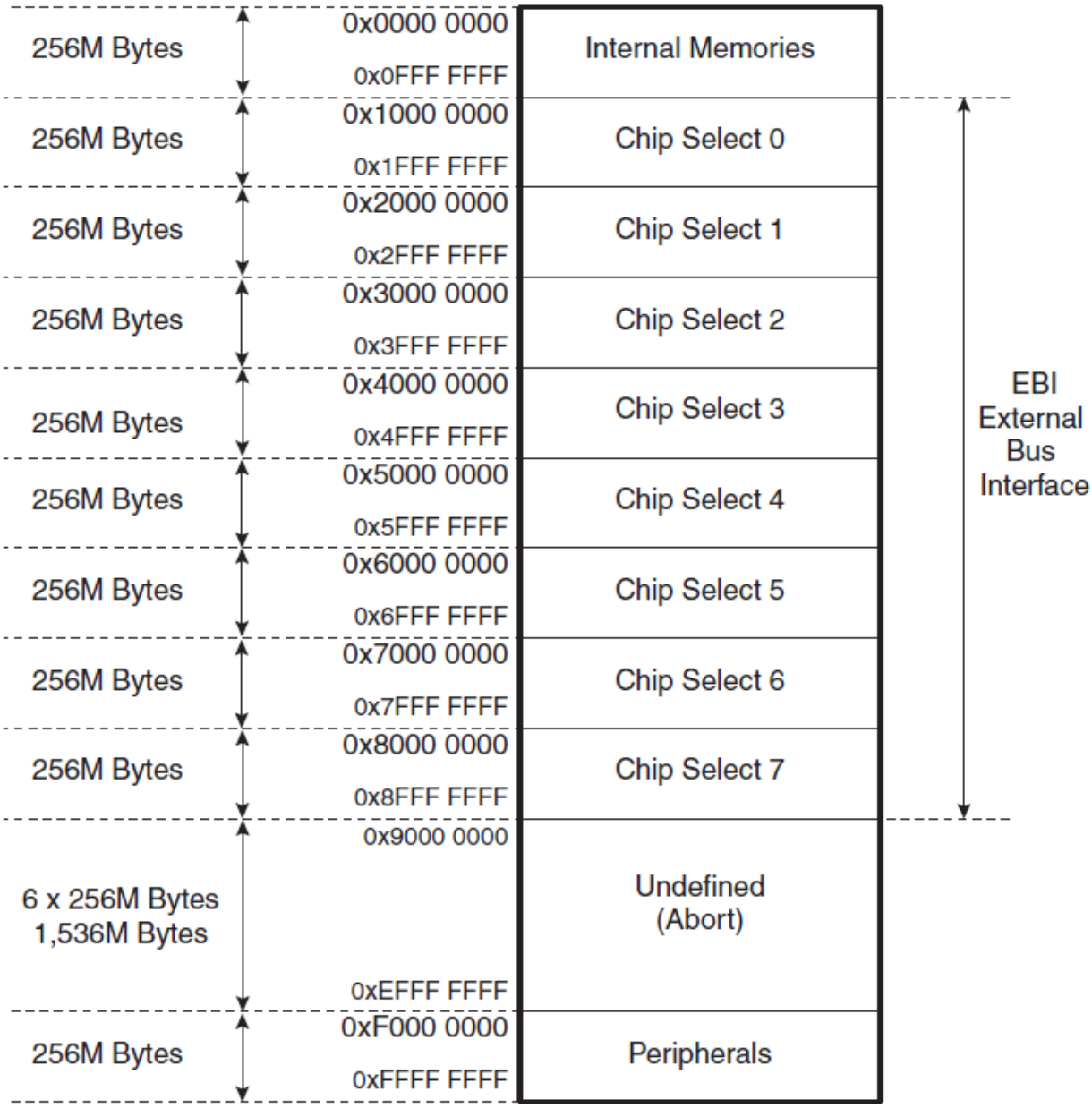
La carte cible possède donc :

- 8 Mo de mémoire Flash : on y mettra le *bootloader u-boot*, le noyau Linux et le système de fichiers *root*.
- 32 Mo de RAM SDRAM.



Carte cible AT91

Le mapping mémoire externe de la carte cible est le suivant :



Mapping mémoire externe de la carte cible AT91

On notera que :

- La mémoire RAM va de $2000 0000 à $21FF FFFF (*Chip Select* 1).
- La mémoire Flash contenant *u-boot* (et éventuellement le noyau Linux et le système de fichiers *root*) va de $1000 0000 à $107F FFFF (*Chip Select* 0).

Le processeur AT91 possède une MMU (*Memory Management Unit*) qui protège les accès mémoire. On utilisera donc le noyau Linux standard pour processor ARM9 de type v4T.

3 TP 0 : PRISE EN MAIN

- Etablir le schéma de l'environnement de développement :
  - Matériels.
  - Liaisons : série, réseau…
  - Logiciels et OS utilisés.
  - Adresses IP du PC de développement (hôte ou *host*) et de la carte cible (cible ou *target*).

- Démarrer le PC sous Linux. Se connecter sous le nom **guest**, mot de passe : **guest** ☺.

- Se créer un répertoire de travail à son nom et s'y placer :
```
host% cd
host% mkdir mon_nom
host% cd mon_nom
```

- Se connecter à la carte AT91 (cible) en utilisant l'outil `minicom` :
```
host% minicom -b 115200
```
Pour sortir de `minicom`, il suffit de taper la combinaison de touches : CTRL A, Z pour accéder au menu et taper q pour quitter.

- Retrouver les commandes du *bootloader u-boot* de la cible pour :
  - Télécharger par le réseau Ethernet via TFTP un fichier binaire `uImage`.
  - Lancer et exécuter le fichier précédemment chargé en mémoire RAM.

- Quelle est l'adresse de point d'entrée de l'exécutable précédemment téléchargé ?

Par la suite, on adoptera les conventions suivantes :
Commande Linux PC hôte pour le développement croisé :
```
host% commande Linux
```
Commande Linux embarqué sur la carte cible AT91 :
```
cpuat91:# commande Linux
```
Commande *u-boot* :
```
CPUAT91=> commande u-boot
```

4 TP 1 : CREATION DU SYSTEME DE FICHIERS ROOT POUR LE NOYAU LINUX STANDARD

Nous allons voir comment créer le système de fichiers *root* qui sera utilisé par le noyau Linux standard exécuté par le processeur de la carte cible AT91. Nous allons aussi y intégrer tous les utilitaires nécessaires pour tester les performances Temps Réel du noyau Linux standard.

- Dans le répertoire à son nom, recopier le fichier `CPUAT91.tgz` sous `~kadionik/` :
```
host% cp /home/kadionik/CPUAT91.tgz .
```

- Décompresser le fichier `CPUAT91.tgz` :
```
host$ tar -xvzf CPUAT91.tgz
```

- Se placer ensuite dans le répertoire `CPUAT91/`. **L'ensemble du travail sera réalisé à partir de ce répertoire ! Les chemins seront donnés par la suite en relatif par rapport à ce répertoire...**
```
host% cd CPUAT91
```

- Créer le système de fichiers *root* squelette `root_fs` pour la carte cible AT91. Que fait le *shell script* `goskel` ?
```
host% cd rootfs
host% ./goskel
```

- Générer les utilitaires de tests `cyclictest`, `stress`... Que fait le *shell script* `go` ? Les installer dans le système de fichiers *root*. Que fait le *shell script* `goinstall` ?
```
host% cd tst
host% cd stress
host% ./go
host% ./goinstall
host% cd tst
host% cd schedutils
host% ./go
host% ./goinstall
host% cd tst
host% cd rt-tests
host% ./go
host% ./goinstall
```

- Compiler `busybox`. Que fait le *shell script* `go` ?
```
host% cd rootfs
host% cd busybox
host% ./go
```

- Générer le système de fichiers *root* final `root_fs` pour la carte cible AT91. Que fait le *shell script* `gorootfs` ?
```
host% cd rootfs
host% ./gorootfs
```

- Quel fichier final génère le *shell script* `gorootfs` ? De quel type est ce fichier ? Le recopier dans le répertoire `/tftpboot/` :
  `host% cp rootfs.jffs2 /tftpboot`

Le système de fichiers JFFS2 (*Journaling Flash File System 2*) est un système de fichiers Linux développé spécialement pour être utilisé sur une mémoire de type Flash que l'on retrouve fréquemment dans un système embarqué. Cela peut être un circuit intégré de mémoire Flash, une carte Compact Flash, une clé USB. La particularité d'une mémoire Flash est sa programmation par secteur (512 octets à quelques Ko) qui nécessite d'effacer complètement le secteur pour y programmer ne serait-ce qu'un seul octet. De plus, il convient de supporter les pannes d'alimentation durant la phase de reprogrammation. JFFS2 est prévu pour cela : c'est un système de fichier journalisé qui stocke toutes les versions d'un fichier modifié, ce qui permet d'accélérer la phase de boot d'un système Linux (pas de `fsck` intempestif) et d'assurer l'intégrité des fichiers stockés. Il intègre un processus de ramasse miettes (*Garbage Collector*) quand le taux d'occupation du système de fichiers est proche de 80-90 % afin de libérer de la place par suppression des trop anciennes versions d'un fichier mais cela a un coût : le temps d'exécution du GC.

La carte cible possède une mémoire Flash mappée dans l'espace d'adressage de $1000 0000 à $1080 0000 exclus.

Le partitionnement de la mémoire Flash est le suivant :

Plage partition	Taille partition	Taille partition	Numéro partition MTD	Nom partition
$107F FFFF $1034 0000	$4C0000 octets	4864 Ko	2	`rootfs`
$1033 FFFF $1004 0000	$300000 octets	3072 Ko	1	`kernel`
$1003 FFFF $1000 0000	$40000 octets	256 Ko	0	`u-boot`

On désire mettre en mémoire Flash le système de fichiers *root*.
L'extrait des traces de boot du noyau données ci-après montrent la détection des 3 partitions de la mémoire Flash de 8 Mo :

```
physmap platform flash device: 01000000 at 10000000
physmap-flash.0: Found 1 x16 devices at 0x0 in 16-bit bank.
Manufacturer ID 0x000089 Chip ID 0x000017
Intel/Sharp Extended Query Table at 0x0031
Intel/Sharp Extended Query Table at 0x0031
Using buffer write method
cfi_cmdset_0001: Erase suspend on write enabled
```

```
3 cmdlinepart partitions found on MTD device physmap-flash.0
Creating 3 MTD partitions on "physmap-flash.0":
0x000000000000-0x000000040000 : "u-boot"
0x000000040000-0x000000340000 : "kernel"
0x000000340000-0x000000800000 : "rootfs"
```

On utilisera la partition MTD 2 `rootfs` pour le système de fichiers *root*.

Pour cela, on utilisera le *bootloader u-boot* pour programmer la mémoire Flash. On partira du fichier `rootfs.jffs2` pour le système de fichiers *root*.

- Prendre la main sur *u-boot* en appuyant sur la touche espace durant le compte à rebours.
- Télécharger puis programmer le système de fichiers *root* `rootfs.jffs2` en mémoire Flash avec *u-boot*. Que fait la fonction *u-boot* `gor` ?
  `CPUAT91=> run gor`

5 TP 2 : COMPILATION DU NOYAU LINUX STANDARD

- Compiler le noyau Linux standard pour la carte cible AT91. Que fait le *shell script* `go` ?
  ```
  host% cd linux-3.14.17
  host% ./go
  ```

- Recopier le fichier du noyau Linux `uImage` dans le répertoire `/tftpboot/` :
  ```
  host% cp arch/arm/boot/uImage /tftpboot
  ```

- Télécharger le fichier `uImage` dans la RAM de la carte cible AT91 avec *u-boot* :
  ```
  CPUAT91=> tftp uImage
  ```

- Lancer le noyau Linux standard :
  ```
  CPUAT91=> bootm
  ```

- Observer les traces de boot du noyau Linux dans la fenêtre `minicom` :

```
Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0x0
Linux version 3.14.17 (guest@linux02) (gcc version 4.8.3 20140320
(prerelease) (Sourcery CodeBench Lite 2014.05-29) )5
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
CPU: VIVT data cache, VIVT instruction cache
Machine: Eukrea
Memory policy: Data cache writeback
AT91: Detected soc type: at91rm9200
AT91: Detected soc subtype: at91rm9200 BGA
AT91: sram at 0x200000 of 0x4000 mapped at 0xfef74000
AT91: filled in soc subtype: at91rm9200 PQFP
Clocks: CPU 179 MHz, master 59 MHz, main 18.432 MHz
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 8128
Kernel command line: root=/dev/mtdblock2 rootfstype=jffs2
console=ttyS0,115200 mtdparts=physmap-flash.0:256k(u-boot),M
PID hash table entries: 128 (order: -3, 512 bytes)
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory: 28192K/32768K available (3020K kernel code, 141K rwdata, 816K
rodata, 115K init, 114K bss, 4576K reserved)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xc2800000 - 0xff000000   ( 968 MB)
    lowmem  : 0xc0000000 - 0xc2000000   (  32 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .text : 0xc0008000 - 0xc03c7254   (3837 kB)
      .init : 0xc03c8000 - 0xc03e4c64   ( 116 kB)
      .data : 0xc03e6000 - 0xc0409780   ( 142 kB)
       .bss : 0xc040978c - 0xc0426308   ( 115 kB)
SLUB: HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
Preemptible hierarchical RCU implementation.
NR_IRQS:16 nr_irqs:16 16
AT91: 96 gpio irqs in 3 banks
sched_clock: 32 bits at 100 Hz, resolution 10000000ns, wraps every
2147483648000000ns
Console: colour dummy device 80x30
Calibrating delay loop... 89.49 BogoMIPS (lpj=447488)
pid_max: default: 32768 minimum: 301
```

```
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x202dd630 - 0x202dd688
pinctrl core: initialized pinctrl subsystem
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
AT91: Power Management
bio: create slab <bio-0> at 0
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
i2c-gpio i2c-gpio.0: using pins 25 (SDA) and 26 (SCL)
Switched to clocksource 32k_counter
NET: Registered protocol family 2
TCP established hash table entries: 1024 (order: 0, 4096 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP: reno registered
UDP hash table entries: 256 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
futex hash table entries: 256 (order: -1, 3072 bytes)
jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat, Inc.
msgmni has been set to 55
io scheduler noop registered
io scheduler deadline registered (default)
atmel_usart.0: ttyS0 at MMIO 0xfffff200 (irq = 17, base_baud = 3744000) is
a ATMEL_SERIAL
console [ttyS0] enabled
atmel_usart atmel_usart.0: Not supported ip name nor version, set to uart
atmel_usart.1: ttyS1 at MMIO 0xfffc0000 (irq = 22, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.1: Not supported ip name nor version, set to uart
atmel_usart.2: ttyS2 at MMIO 0xfffc4000 (irq = 23, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.2: Not supported ip name nor version, set to uart
atmel_usart.3: ttyS3 at MMIO 0xfffc8000 (irq = 24, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.3: Not supported ip name nor version, set to uart
atmel_usart.4: ttyS4 at MMIO 0xfffcc000 (irq = 25, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.4: Not supported ip name nor version, set to uart
brd: module loaded
loop: module loaded
nbd: registered device at major 43
physmap platform flash device: 01000000 at 10000000
physmap-flash.0: Found 1 x16 devices at 0x0 in 16-bit bank. Manufacturer ID
0x000089 Chip ID 0x000017
Intel/Sharp Extended Query Table at 0x0031
Intel/Sharp Extended Query Table at 0x0031
Using buffer write method
cfi_cmdset_0001: Erase suspend on write enabled
3 cmdlinepart partitions found on MTD device physmap-flash.0
Creating 3 MTD partitions on "physmap-flash.0":
0x000000000000-0x000000040000 : "u-boot"
0x000000040000-0x000000340000 : "kernel"
0x000000340000-0x000000800000 : "rootfs"
mtd-ram mtd-ram.0: registered mtd device
libphy: MACB_mii_bus: probed
```

```
at91_ether   at91_ether   eth0:   attached   PHY   driver   [Generic   PHY]
(mii_bus:phy_addr=at91_ether-fffff:01, irq=-1)
at91_ether   at91_ether   eth0:   AT91   ethernet   at   0xfffbc000   int=40
(0a:03:4a:c6:f3:a2)
PPP generic driver version 2.4.2
PPP BSD Compression module registered
PPP Deflate Compression module registered
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
ohci-atmel: OHCI Atmel driver
at91_ohci at91_ohci: OHCI Host Controller
at91_ohci at91_ohci: new USB bus registered, assigned bus number 1
at91_ohci at91_ohci: irq 39, io mem 0x00300000
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
usbcore: registered new interface driver usb-storage
udc: at91_udc version 3 May 2006
using random self ethernet address
using random host ethernet address
usb0: HOST MAC ce:a5:81:f0:3d:43
usb0: MAC 42:b1:b9:8c:3c:25
using random self ethernet address
using random host ethernet address
g_ether gadget: Ethernet Gadget, version: Memorial Day 2008
g_ether gadget: g_ether ready
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
at91rm9200_wdt: AT91 Watchdog Timer enabled (5 seconds, nowayout)
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
VFS: Mounted root (jffs2 filesystem) on device 31:2.
Freeing unused kernel memory: 112K (c03c8000 - c03e4000)
    Hostname       : cpuat91
    Kernel release : Linux 3.14.17
    Kernel version : #4 PREEMPT Tue Jan 13 14:48:16 CET 2015

 Mounting /proc             : [SUCCESS]
 Mounting /sys              : [SUCCESS]
 Mounting /dev              : [SUCCESS]
 Mounting /dev/pts          : [SUCCESS]
 Enabling hot-plug          : [SUCCESS]
 Populating /dev            : [SUCCESS]
 Mounting other filesystems : [SUCCESS]
 Starting telnetd           : [SUCCESS]
 Network configuration      : [SUCCESS]

System initialization complete.

Please press Enter to activate this console.

cpuat91:/#
cpuat91:/# ps
PID   USER     TIME   COMMAND
    1 root       0:02 init
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
    4 root       0:00 [kworker/0:0]
    5 root       0:00 [kworker/0:0H]
    6 root       0:00 [kworker/u2:0]
    7 root       0:00 [rcu_preempt]
    8 root       0:00 [rcu_sched]
    9 root       0:00 [rcu_bh]
   10 root       0:00 [khelper]
   11 root       0:00 [kworker/u2:1]
  149 root       0:00 [writeback]
```

```
  152 root       0:00 [bioset]
  153 root       0:00 [kblockd]
  165 root       0:00 [khubd]
  260 root       0:00 [kworker/0:1]
  261 root       0:00 [rpciod]
  268 root       0:00 [kswapd0]
  317 root       0:00 [fsnotify_mark]
  326 root       0:00 [nfsiod]
  564 root       0:00 [deferwq]
  565 root       0:00 [jffs2_gcd_mtd2]
  581 root       0:00 /usr/sbin/telnetd
  585 root       0:00 -/bin/ash
  586 root       0:00 ps
cpuat91:/#
```

6 TP 3 : MESURE DE TEMPS DE LATENCE AVEC LE NOYAU LINUX STANDARD

Nous allons mesurer des temps de latence du noyau Linux standard dans le cas d'un noyau non stressé puis dans le cas d'un noyau stressé.

Pour stresser le noyau, on utilisera l'utilitaire `stress`.

Pour mesurer les temps de latence, on utilisera l'utilitaire `cyclictest`.

- Dévalider le *throttling* :
  ```
  cpuat91:# echo -1 > /proc/sys/kernel/sched_rt_runtime_us
  ```

Noyau standard non stressé :

- Lancer `cyclictest`. Noter le temps de latence maximum au bout de 5 minutes de tests :
  ```
  cpuat91:# cyclictest -n -p 99 -i 5000
  ```

Noyau standard stressé :

- Stresser le noyau avec `stress`. Que fait le programme `stress` ?
  ```
  cpuat91:# stress -c 50 -i 50 &
  ```

- Lancer `cyclictest`. Noter le temps de latence maximum au bout de 5 minutes de tests :
  ```
  cpuat91:# cyclictest -n -p 99 -i 5000
  ```

7 TP 4 : CREATION DU SYSTEME DE FICHIERS ROOT POUR LE NOYAU LINUX XENOMAI

Nous allons voir comment créer le système de fichiers *root* qui sera utilisé par le noyau Linux Xenomai exécuté par le processeur de la carte cible AT91. Nous allons aussi y intégrer tous les utilitaires nécessaires pour tester les performances Temps Réel du noyau Linux Xenomai.

- Créer le système de fichiers *root* squelette `root_fs` pour la carte cible AT91 :
```
host% cd rootfs
host% ./goskel
```

- Générer les utilitaires de tests `cyclictest`, `stress`...
```
host% cd tst
host% cd stress
host% ./goinstall
host% cd tst
host% cd schedutils
host% ./goinstall
host% cd tst
host% cd rt-tests
host% ./goinstall
```

- Générer les utilitaires de tests de Xenomai `cyclictest`, `latency`... Que fait le *shell script* `goconfig` ? Que fait le *shell script* `go` ?
```
host% cd xenomai
host% ./goconfig
host% ./go
```

- Générer le système de fichiers *root* final `root_fs` pour la carte cible AT91. Que fait le *shell script* `gorootfsxenomai` ? Il est demandé à un moment donné de rentrer le mot de passe de guest (`[sudo] password for guest :`) pour créer des points d'entrée de périphériques dans le système de fichiers *root* :
```
host% cd rootfs
host% ./gorootfsxenomai
```

- Quel fichier final génère le *shell script* `gorootfsxenomai` ? De quel type est ce fichier ? Le recopier dans le répertoire `/tftpboot/` :
```
host% cp rootfs.jffs2 /tftpboot
```

- Prendre la main sur *u-boot* en appuyant sur la touche espace durant le compte à rebours.

- Télécharger puis programmer le système de fichiers *root* `rootfs.jffs2` en mémoire Flash avec *u-boot* :
```
CPUAT91=> run gor
```

8 TP 5 : COMPILATION DU NOYAU LINUX XENOMAI

- Appliquer le patch Xenomai sur le noyau Linux. Que fait le *shell script* `go-ipipe` ?
```
host% cd xenomai
host% ./go-ipipe
```

- Compiler le noyau Linux standard pour la carte cible AT91. Que fait le *shell script* `go` ?
```
host% cd linux-3.14.17-xenomai
host% ./go
```

- Recopier le fichier du noyau Linux `uImage` dans le répertoire `/tftpboot/` :
```
host% cp arch/arm/boot/uImage /tftpboot
```

- Télécharger le fichier `uImage` dans la RAM de la carte cible AT91 avec *u-boot* :
```
CPUAT91=> tftp uImage
```

- Lancer le noyau Linux standard :
```
CPUAT91=> bootm
```

- Observer les traces de boot du noyau Linux dans la fenêtre `minicom` :

```
Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0x0
Linux version 3.14.17-xenomai (guest@linux02) (gcc version 4.8.3 20140320
(prerelease) (Sourcery CodeBench Lite 2014.5
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
CPU: VIVT data cache, VIVT instruction cache
Machine: Eukrea
Memory policy: Data cache writeback
AT91: Detected soc type: at91rm9200
AT91: Detected soc subtype: at91rm9200 BGA
AT91: sram at 0x200000 of 0x4000 mapped at 0xfef74000
AT91: filled in soc subtype: at91rm9200 PQFP
Clocks: CPU 179 MHz, master 59 MHz, main 18.432 MHz
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 8128
Kernel      command      line:      root=/dev/mtdblock2      rootfstype=jffs2
console=ttyS0,115200 mtdparts=physmap-flash.0:256k(u-boot),M
PID hash table entries: 128 (order: -3, 512 bytes)
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory:  27664K/32768K  available  (3317K  kernel  code,  159K  rwdata,  872K
rodata, 121K init, 267K bss, 5104K reserved)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xc2800000 - 0xff000000   ( 968 MB)
    lowmem  : 0xc0000000 - 0xc2000000   (  32 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .text : 0xc0008000 - 0xc041f8dc   (4191 kB)
      .init : 0xc0420000 - 0xc043e428   ( 122 kB)
      .data : 0xc0440000 - 0xc0467f20   ( 160 kB)
       .bss : 0xc0467f2c - 0xc04aac08   ( 268 kB)
SLUB: HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
Preemptible hierarchical RCU implementation.
NR_IRQS:16 nr_irqs:16 16
AT91: 96 gpio irqs in 3 banks
```

```
Interrupt pipeline (release #4)
sched_clock: 32 bits at 100 Hz, resolution 10000000ns, wraps every
2147483648000000ns
Console: colour dummy device 80x30
Calibrating delay loop... 89.49 BogoMIPS (lpj=447488)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x20325ba8 - 0x20325c00
pinctrl core: initialized pinctrl subsystem
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
AT91: Power Management
AT91 I-pipe timer: using TC0, div: 32, freq: 1.872000 MHz
I-pipe, 1.872 MHz clocksource, wrap in 35 ms
bio: create slab <bio-0> at 0
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
i2c-gpio i2c-gpio.0: using pins 25 (SDA) and 26 (SCL)
Switched to clocksource ipipe_tsc
NET: Registered protocol family 2
TCP established hash table entries: 1024 (order: 0, 4096 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP: reno registered
UDP hash table entries: 256 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
futex hash table entries: 256 (order: -1, 3072 bytes)
I-pipe: head domain Xenomai registered.
Xenomai: hal/arm started.
Xenomai: scheduling class idle registered.
Xenomai: scheduling class rt registered.
Xenomai: real-time nucleus v2.6.4 (Jumpin' Out) loaded.
Xenomai: debug mode enabled.
Xenomai: starting native API services.
Xenomai: starting POSIX services.
Xenomai: starting RTDM services.
jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat, Inc.
msgmni has been set to 54
io scheduler noop registered
io scheduler deadline registered (default)
atmel_usart.0: ttyS0 at MMIO 0xfffff200 (irq = 17, base_baud = 3744000) is
a ATMEL_SERIAL
console [ttyS0] enabled
atmel_usart atmel_usart.0: Not supported ip name nor version, set to uart
atmel_usart.1: ttyS1 at MMIO 0xfffc0000 (irq = 22, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.1: Not supported ip name nor version, set to uart
atmel_usart.2: ttyS2 at MMIO 0xfffc4000 (irq = 23, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.2: Not supported ip name nor version, set to uart
atmel_usart.3: ttyS3 at MMIO 0xfffc8000 (irq = 24, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.3: Not supported ip name nor version, set to uart
atmel_usart.4: ttyS4 at MMIO 0xfffcc000 (irq = 25, base_baud = 3744000) is
a ATMEL_SERIAL
atmel_usart atmel_usart.4: Not supported ip name nor version, set to uart
```

```
brd: module loaded
loop: module loaded
nbd: registered device at major 43
physmap platform flash device: 01000000 at 10000000
physmap-flash.0: Found 1 x16 devices at 0x0 in 16-bit bank. Manufacturer ID
0x000089 Chip ID 0x000017
Intel/Sharp Extended Query Table at 0x0031
Intel/Sharp Extended Query Table at 0x0031
Using buffer write method
cfi_cmdset_0001: Erase suspend on write enabled
3 cmdlinepart partitions found on MTD device physmap-flash.0
Creating 3 MTD partitions on "physmap-flash.0":
0x000000000000-0x000000040000 : "u-boot"
0x000000040000-0x000000340000 : "kernel"
0x000000340000-0x000000800000 : "rootfs"
mtd-ram mtd-ram.0: registered mtd device
libphy: MACB_mii_bus: probed
at91_ether   at91_ether   eth0:   attached   PHY   driver   [Generic   PHY]
(mii_bus:phy_addr=at91_ether-fffff:01, irq=-1)
at91_ether   at91_ether   eth0:   AT91   ethernet   at   0xfffbc000   int=40
(0a:03:4a:c6:f3:a2)
PPP generic driver version 2.4.2
PPP BSD Compression module registered
PPP Deflate Compression module registered
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
ohci-atmel: OHCI Atmel driver
at91_ohci at91_ohci: OHCI Host Controller
at91_ohci at91_ohci: new USB bus registered, assigned bus number 1
at91_ohci at91_ohci: irq 39, io mem 0x00300000
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
usbcore: registered new interface driver usb-storage
udc: at91_udc version 3 May 2006
using random self ethernet address
using random host ethernet address
usb0: HOST MAC 76:e9:75:52:99:7b
usb0: MAC 36:77:bd:cd:bd:24
using random self ethernet address
using random host ethernet address
g_ether gadget: Ethernet Gadget, version: Memorial Day 2008
g_ether gadget: g_ether ready
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
at91rm9200_wdt: AT91 Watchdog Timer enabled (5 seconds, nowayout)
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
VFS: Mounted root (jffs2 filesystem) on device 31:2.
Freeing unused kernel memory: 120K (c0420000 - c043e000)
    Hostname       : cpuat91
    Kernel release : Linux 3.14.17-xenomai
    Kernel version : #7 PREEMPT Tue Jan 13 16:04:10 CET 2015

 Mounting /proc             : [SUCCESS]
 Mounting /sys              : [SUCCESS]
 Mounting /dev              : [SUCCESS]
 Mounting /dev/pts          : [SUCCESS]
 Enabling hot-plug          : [SUCCESS]
 Populating /dev            : [SUCCESS]
 Mounting other filesystems : [SUCCESS]
 Starting telnetd           : [SUCCESS]
 Network configuration      : [SUCCESS]

System initialization complete.
```

```
Please press Enter to activate this console.

cpuat91:/#
cpuat91:/# ps
PID   USER     TIME   COMMAND
    1 root       0:02 init
    2 root       0:00 [kthreadd]
    3 root       0:00 [ksoftirqd/0]
    4 root       0:00 [kworker/0:0]
    5 root       0:00 [kworker/0:0H]
    6 root       0:00 [kworker/u2:0]
    7 root       0:00 [rcu_preempt]
    8 root       0:00 [rcu_sched]
    9 root       0:00 [rcu_bh]
   10 root       0:00 [khelper]
   11 root       0:00 [kworker/u2:1]
  167 root       0:00 [writeback]
  170 root       0:00 [bioset]
  171 root       0:00 [kblockd]
  183 root       0:00 [khubd]
  278 root       0:00 [kworker/0:1]
  279 root       0:00 [rpciod]
  319 root       0:00 [gatekeeper/0]
  321 root       0:00 [kswapd0]
  370 root       0:00 [fsnotify_mark]
  379 root       0:00 [nfsiod]
  617 root       0:00 [deferwq]
  618 root       0:00 [jffs2_gcd_mtd2]
  634 root       0:00 /usr/sbin/telnetd
  638 root       0:00 -/bin/ash
  639 root       0:00 ps
cpuat91:/#
```

9 TP 6 : MESURE DE TEMPS DE LATENCE AVEC LE NOYAU LINUX XENOMAI

Nous allons mesurer des temps de latence sur le noyau Linux Xenomai dans le cas d'un noyau non stressé puis dans le cas d'un noyau stressé.

Pour stresser le noyau, on utilisera l'utilitaire `stress`.

- Dévalider le *throttling* :
  ```
  cpuat91:# echo -1 > /proc/sys/kernel/sched_rt_runtime_us
  ```

Noyau Xenomai non stressé. Outils standards :

- Lancer `cyclictest`. Noter le temps de latence maximum au bout de 5 minutes de tests :
  ```
  cpuat91:# cyclictest -n -p 99 -i 5000
  ```

Noyau Xenomai stressé. Outils standards :

- Stresser le noyau avec `stress` :
  ```
  cpuat91:# stress -c 50 -i 50 &
  ```

- Lancer `cyclictest`. Noter le temps de latence maximum au bout de 5 minutes de tests :
  ```
  cpuat91:# cyclictest -n -p 99 -i 5000
  ```

Noyau Xenomai non stressé. Outils Xenomai :

On utilisera maintenant les outils Xenomai qui se trouvent dans le répertoire `/usr/xenomai/bin/`.

- Lancer l'outil Xenomai `cyclictest`. Noter le temps de latence maximum au bout de 5 minutes de tests :
```
cpuat91:# /usr/xenomai/bin/cyclictest –n –p 99 –i 5000
```

- On utilise maintenant l'outil Xenomai `latency` dans 3 modes différents. A quoi correspondent ces 3 modes ? Noter pour les 3 modes le temps de latence maximum au bout de 5 minutes de tests :
```
cpuat91:# /usr/xenomai/bin/latency –t0 –p 5000
cpuat91:# /usr/xenomai/bin/latency –t1 –p 5000
cpuat91:# /usr/xenomai/bin/latency –t2 –p 5000
```

Noyau Xenomai stressé. Outils Xenomai :

- Stresser le noyau avec `stress` :
```
cpuat91:# stress –c 50 –i 50 &
```

- Lancer l'outil Xenomai `cyclictest`. Noter le temps de latence maximum au bout de 5 minutes de tests :
```
cpuat91:# /usr/xenomai/bin/cyclictest –n –p 99 –i 5000
```

- Lancer l'outil Xenomai `latency` dans les 3 modes. Noter pour les 3 modes le temps de latence maximum au bout de 5 minutes de tests :
```
cpuat91:# /usr/xenomai/bin/latency –t0 –p 5000
cpuat91:# /usr/xenomai/bin/latency –t1 –p 5000
cpuat91:# /usr/xenomai/bin/latency –t2 –p 5000
```

10 CONCLUSION

On complètera le tableau suivant avec les mesures de temps de latence pour une comparaison facile :

Temps de latence en μs	**Linux standard non stressé**	**Linux standard stressé**	**Linux Xenomai non stressé**	**Linux Xenomai stressé**
`cyclictest`				
`cyclictest Xenomai`				
`latency -t0`				
`latency -t1`				
`latency -t2`				

Que peut-on en conclure si l'on compare les résultats obtenus avec le noyau standard avec ceux obtenus avec le noyau Xenomai ?

Une mesure pendant juste 5 minutes est-elle suffisante ? Les conditions de stress imposées au noyau sont-elles suffisantes ?

REFERENCES

- Cours ENSEIRB de Systèmes embarqués, Linux embarqué. P. Kadionik.
- Linux embarqué. P. Ficheux. Editions Eyrolles.
- AT91RM9200 Datasheet. Atmel.
- CPUAT91. Documentation Technique. EUKREA Electromatique.
- CPUAT91. Guide de démarrage rapide. EUKREA Electromatique.

11 ANNEXE 1 : CONFIGURATION RESEAU HOTES ET CIBLES

	POSTE PC01	
	NOM	**ADRESSE IP**
HOTE	rodirula	10.7.4.223
CIBLE	blackfin001	10.7.2.118

	POSTE PC02	
	NOM	**ADRESSE IP**
HOTE	sarracenia	10.7.4.225
CIBLE	blackfin002	10.7.2.119

	POSTE PC03	
	NOM	**ADRESSE IP**
HOTE	utricularia	10.7.4.227
CIBLE	blackfin003	10.7.2.120

	POSTE PC04	
	NOM	**ADRESSE IP**
HOTE	ibicella	10.7.2.112
CIBLE	blackfin004	10.7.2.121

	POSTE PC05	
	NOM	**ADRESSE IP**
HOTE	nepenthes	10.7.2.114
CIBLE	blackfin005	10.7.2.122

	POSTE PC06	
	NOM	**ADRESSE IP**
HOTE	pinguicula	10.7.2.116
CIBLE	blackfin006	10.7.2.123

Masque de sous réseau : **255.255.248.0**

Exemple : configuration réseau de la carte cible `blackfin001` :

```
target# ifconfig eth0 10.7.2.118 netmask 255.255.248.0
```