Linux embarqué sur APF27

Pierre-Yves Duval

Ecole d'informatique temps réel IN2P3: Fréjus 23-26 Novembre 2009

Généralités: pourquoi Linux

Objectif:

Avoir un système permettant une grande palette d'applications dans un environnement système bien connu supporté par une très large communauté ouverte et participative.

Pouvoir bénéficier au maximum des logiciels libres et des développements communautaires en facilitant leur portage sur notre plateforme d'exécution.

Solution: distribution Linux embarqué (noyau + applications):

Le noyau Linux est modulaire et peut -être décliné/taillé avec des versions occupant des volumes réduits compatibles avec les limites des plateformes embarquées d'aujourd'hui mais il implique:

une machine 32 bits (mini) et une MMU (sinon µClinux)

Le nombre d'applications peut être limité au strict minimum nécessaire sur une plateforme embarquée et des versions allégées existent pour la plupart des outils sous Linux.

Composants de Linux embarqué

Une distribution Linux c'est:

- un **noyau** qui est chargé en mémoire au moment du boot et y demeure

- un **root file system** (rootfs) système de fichiers qui contient:
    - les modules drivers chargés dynamiquement (/lib ...)
    - les fichiers de configurations du système (/etc ...)
    - les exécutables des outils standards (shell, réseau, ...)
    - des applications (/opt /usr ...)

Minimum: /dev, /proc, /bin, /etc, /lib, /usr, /tmp,

La séquence de boot

Linux est démarré par un **boot loader** (exécutable présent sur la carte) U-boot pour ARMADEUS qui est aussi une sorte de « BIOS » ou « moniteur interactif »:

1 - récupère le noyau Linux et le root file system (rootfs racine /) en local (flash ou MMC) ou via le réseau.

2 - il charge le code du noyau Linux avec un système de fichiers minimaliste en RAM et lance son exécution

3 - lorsque les initialisations minimum du hardware ont été faites, Linux peut charger/monter son vrai rootfs et lancer sa séquence d'initialisation complète avec tous les drivers.

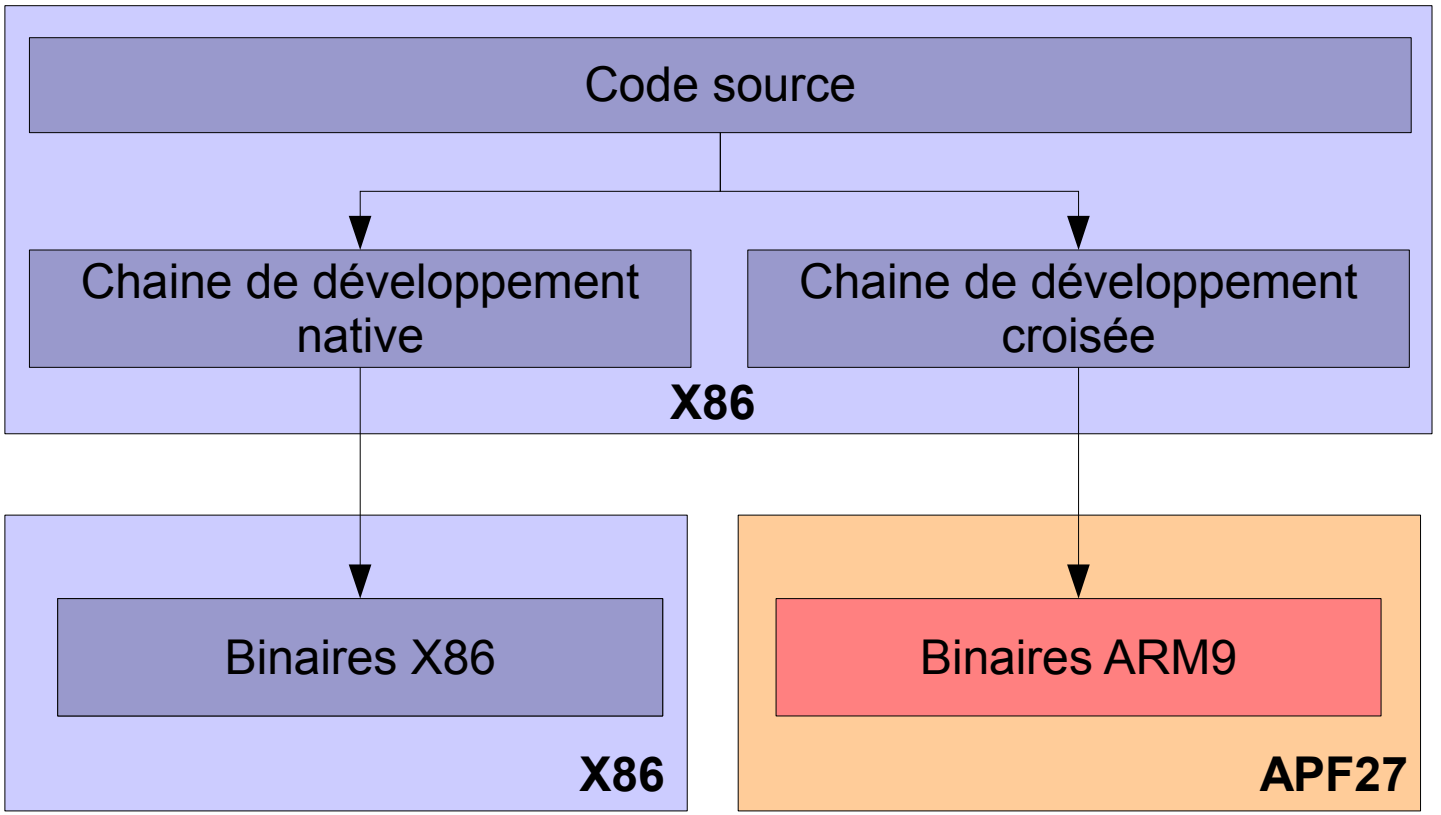
dans le cas de l'APF27 le système de fichier est un RAMFS système de fichier tout en mémoire car il n'y a pas de disque connecté.

Développement

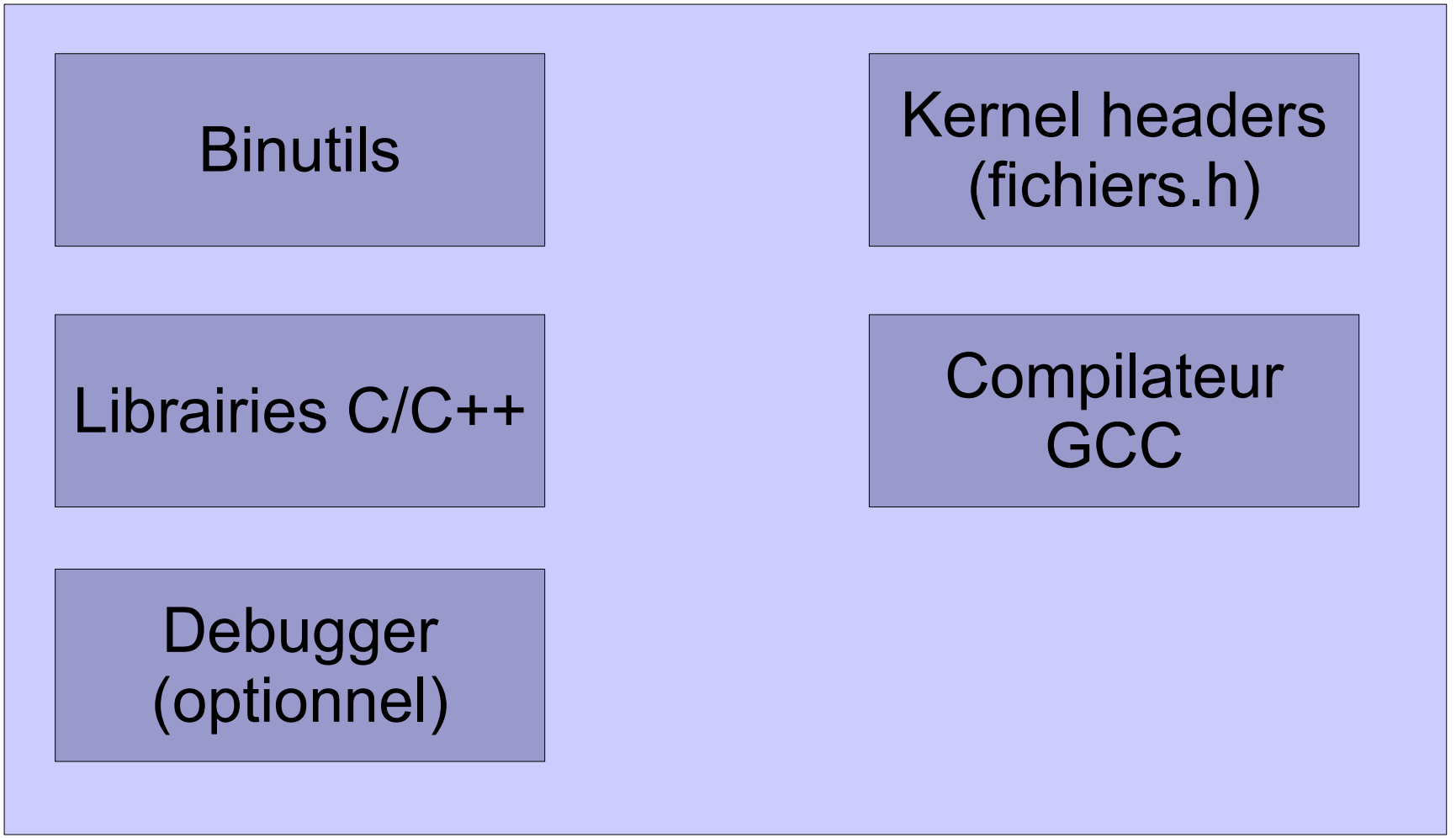
On ne peut développer en natif sur une plateforme embarquée si elle est assez puissante

mais

On put utiliser une autre machine (station de travail avec un bel écran et tout et tout) pour faire du développement croisé et ensuite exécuter le code produit sur la plateforme cible.



Composants pour le développement



Binutils

Ensemble d'outils logiciels pour générer et manipuler les binaires pour une architecture Particulière (X86, NIOS2, **ARM**, MIPS ...)

asm	l'assembleur du processeur
ld	l'éditeur de lien
ar, ranlib	le générateur d'archive de code exécutable
objdump, readelf, size, mn, strings	Outils d'inspection du code généré
strip	réducteur de librairie pour ne garder que les fonctions utilisées

Voir http://www.gnu.org/software/binutils/

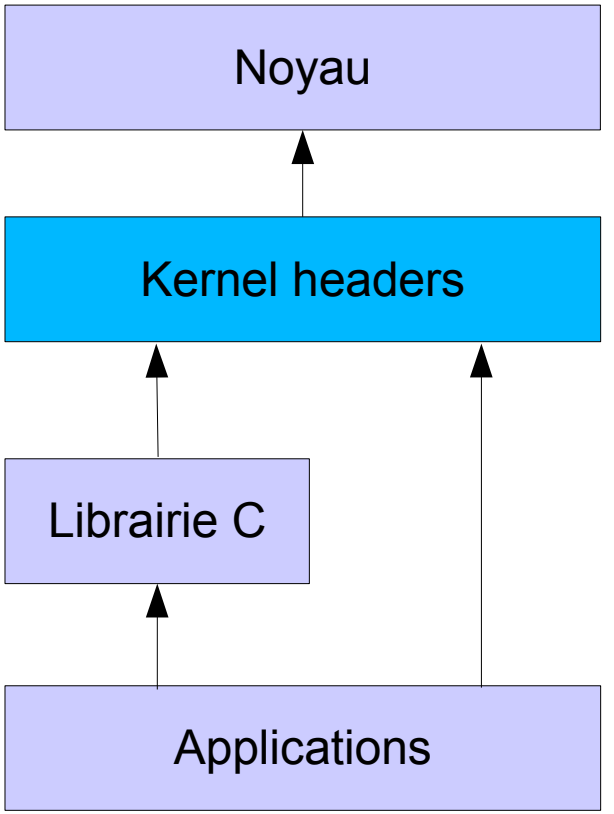
Kernel headers

La librairie C et les applications utilisent/appellent
les fonctions du noyau, elles ont besoin d'avoir accès:

- aux fonctions système (signature)
- constantes utilisées par le système
- structures de données propres au système

La librairie C encapsule de nombreux appels au système, certaines applications peuvent appeler directement le Système. Elles ont donc besoin de connaître des définitions de ces interface disponibles dans les fichier.h du système.

Examples:
<linux/...>
<asm/...>
Sous le répertoire include/ des sources du système.



Compilateur et librairie C

GCC c'est le compilateur GNU associé à Linux

Peut compiler différents langages (C, C++, Ada, Fortran, Java, ObjectiveC,ObjectiveC++) pour une variété de plateformes (NIOS2, **ARM**, MIPS, X86 etc ...)

La librairie C est l'outil principal d'appel au système. Il en existe plusieurs versions: glibc (la plus complète mais volumineuse), uClibc,eglibc, dietlibc, newlibc etc ...

Pour le l'APF27 c'est uClibc qui est utilsée
elle plus compacte que la glibc.

Attention: la même librairie doit être utilisée pour construire la chaîne de développement et les applications

Exemple en compilation statique

	glibc	uClibc
« hello world »	475K	25k
busybox	843K	311K

Construction de Linux pour APF27

Pour construire une chaine de construction « buildtool » et Linux il faut (entre autre):

- Récupérer les header du noyau
- Récupérer, configurer, compiler et installer les binutils
- Récupérer, configurer, compiler et installer une première version de GCC pour cross compiler les librairies
- Reconfiguration et recompilation de GCC pour la cible qu'on utilise avec les librairies.
- Récupérer les sources, configurer et construire Linux

Ces opérations sont complexes et pénibles.

Des chaînes de développement ont été créées pour automatiser toutes ces étapes.
L'outil open source qui a été sélectionné pour APF27/ARM9 est

Buildroot

Pour Linux sur APF27 un TWIKI est accessible qui documente toutes les étapes de construction avec un buildroot adapté par ARMADEUS pour l'apf27.

http://www.armadeus.com/wiki/index.php?title=Setup

On récupère non seulement le noyau Linux standard pour ARM9 mais aussi les drivers spécifiques à la plateforme cible : i.MX27(freescale) et APF27(ARMADEUS).

Construction de Linux pour APF27

1- Il faut s'assurer d'avoir tous les outils de développement standards sous sa machine Linux:
(gcc gcc-c++ make autoconf automake libtool bison flex gettext patch subversion texinfo git ...)

2- Récupère le tar file d'armadeus avec l'arborescence de construction
tar xjvf armadeus-3.1.tar.bz2
Ou on peu récupèrer la dernière version du projet en cours de dévoppement chez armadeus avec git
git clone git://armadeus.git.sourceforge.net/gitroot/armadeus/armadeus armadeus

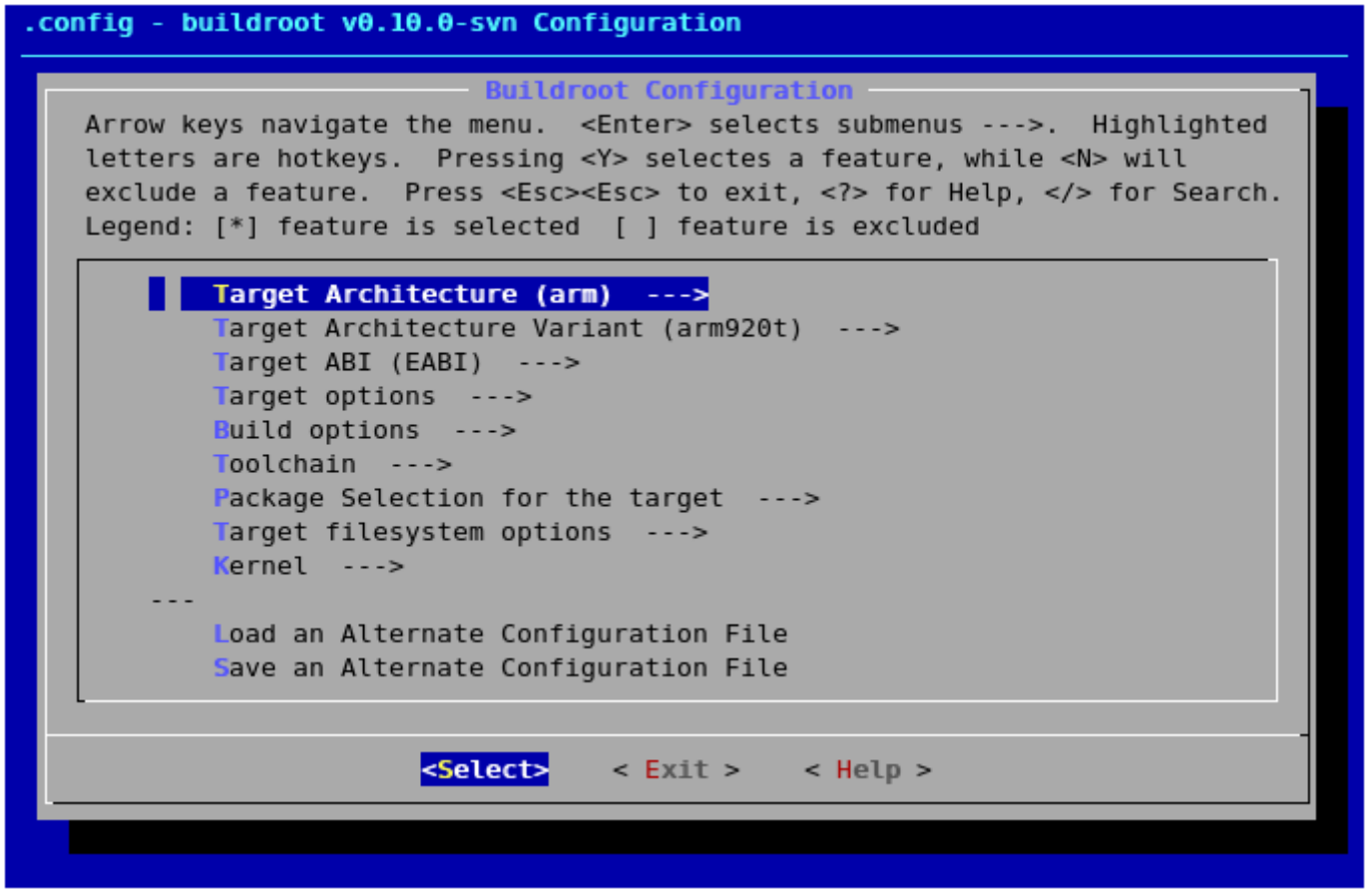
3- Et on construit dans le répertoire racine « armadeus »
La commande **make apf27_defconfig** permet de fixer les paramètres de construction:
- Construction du noyau (processeur, taille mémoire ...)
- Type de systèmes de fichiers supportés
- Driver a inclure ou pas
- Applications/Packages à inclure ou pas

une configuration par défaut est proposée qu'il suffit de modifier/compléter

4-Puis la commande **make** permet de lancer toute la construction buildtools + Linux.

5- Résultats:
- Cross-compilateur et librairies
- Les fichiers noyaux, root file system compressé et moniteur U-boot

Construction de Linux pour APF27



Les répertoires de construction

armadeus/
  buildroot/
    binaries/apf27/ (binaires à charger noyaux, rootfs, U-boot)
    build_armv5te/ (sources non configurables python, zlib, strace ...)
    project_build_armv5te/apf27/ (sources configurables busybox, linux, u-boot ...)
    project_build_armv5te/stagging_dir/usr/bin (exécutables compilateurs et binutils)
    project_build_armv5te/stagging_dir/lib (librairies dont uClibc)
    toolchain_build_armv5te/ (source de la chaine de cross compilation)
    target/device/aramdeus/rootfs/target_skeleton (squelette du rootfs a produire **modifiable**)

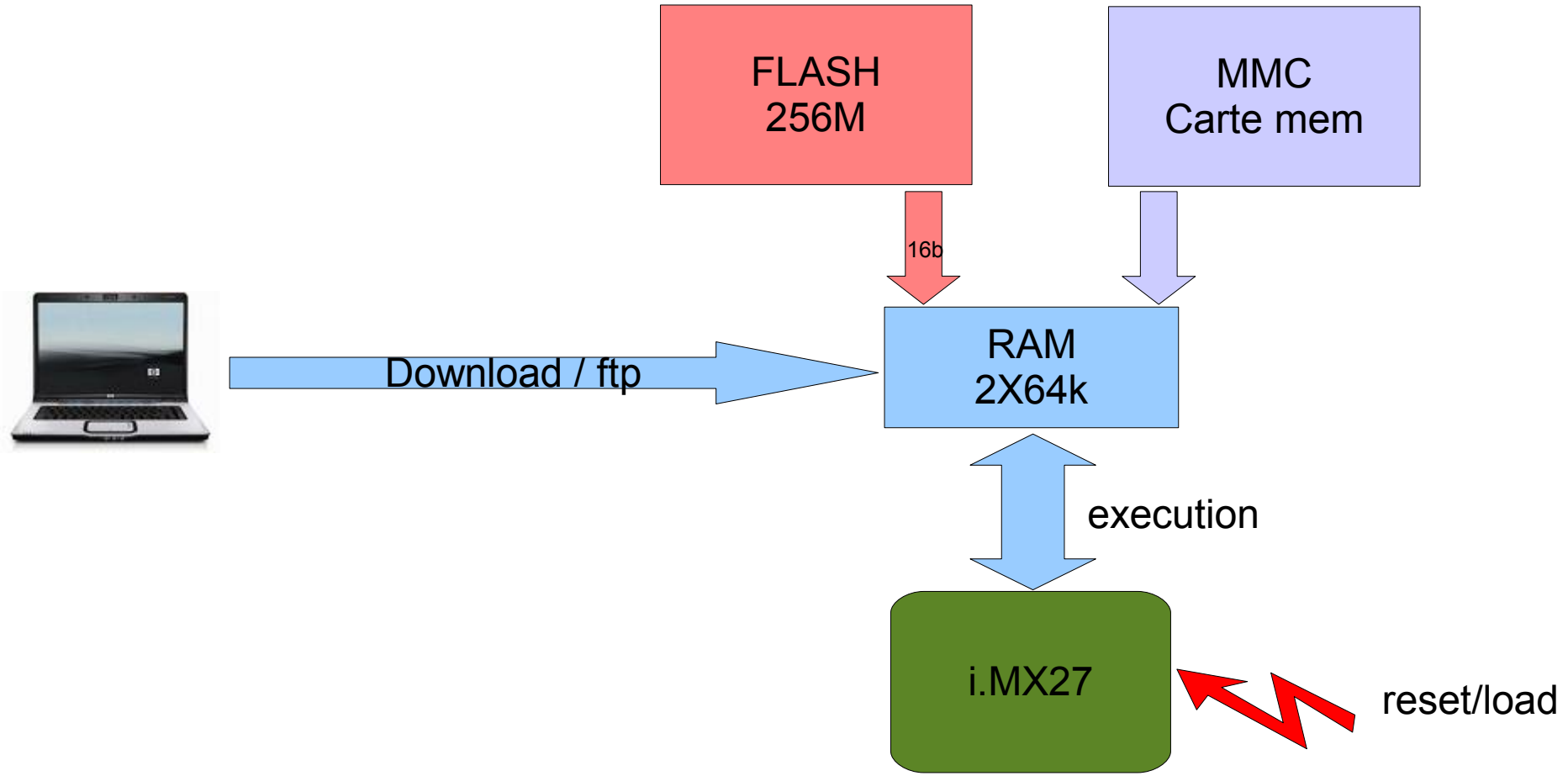
...

  target/linux/modules (sources et binaires drivers sous forme de modules chargeables Linux)

Les fichiers systèmes produits par la construction sont:

- apf27-linux.bin (le noyau linux bootable)
- rootfs.arm.jffs2 (image du FileSystem/RootFS a utiliser avec U-Boot)
- rootfs.arm.tar (autre format du FileSystem/RootFS utilisable pour le boote NFS ou MMC (carte mémoire))
- u-boot.bin (le binaire de u-boot)

Mémoires utilisées



Initialisation/configuration

Lancer une application au démarrage (rootfs): créer un fichier /etc/init.d/S99app très simple

```
#!/bin/sh
/usr/bin/monApplication &
Exit 0
```

Paramètres réseau de la carte (via U-boot):

BIOS> **setenv** netmask 255.255.255.0
BIOS> **setenv** ipaddr 192.168.0.10
BIOS> **setenv** serverip 192.168.0.2 (adresse du server/PC contenant les fichiers à télécharger)
BIOS> **setenv** rootpath "/tftpboot" (pour booter sur le file system Linux via NFS)
BIOS> **saveenv** (mémorise ces paramètres en flash)

Peuvent aussi aussi être mis via un serveur dhcp si on boote de cette façon

Différentes façons de travailler

Quand l'application est au point: faire une release et flasher:

1 - l'intégrer comme application dans l'arbre de construction buildroot
2 - écrire le fichier /etc/init.d de lancement de celle-ci dès la fin du boot
3 - générer le noyau + rootfs + Uboot qu'on copie dans /tftpboot
4 - paramétriser les valeurs de l'application sous U-boot: **setenv** para valeur
5 - les sauvegarder: **saveenv**
(autre solution via un serveur dhcp sur lequel on déclare ces valeurs pour un boot dhcp)
6 - flasher le noyau dans la cible **run** update_kernel
7 - flasher le rootfs dans la cible **run** update_rootfs

Au prochain reset de la carte:
- elle bootera
- elle lancera les applications de service listées dans /etc/init.d

Différentes façons de travailler

Interfaces de commande:

Console: via la liaison série

Terminaux: possibilité par telnet d'avoir d'autres fenêtres, utile en exploitation

Noyau:

Le noyau doit être (re)téléchargé par la liason série sous U-boot après chaque modification (inclus les drivers intégrés)

Copier le noyau dans /tftpboot

BIOS> **setenv** rootpath "/tftpboot "

BIOS> **tftpboot** ${loadaddr} apf27-linux.bin (charge le noyau en mémoire RAM et boote dessus)

Différentes façons de travailler

Développement facile avec NFS:

Monter un répertoire de développement sur la cible permet de profiter d'un l'environnement de travail confortable sur la station et d'avoir les binaires à tester immédiatement accessibles (driver sous forme de modules chargeables inclus)

- A la main: **mount** -t nfs 192.168.0.2:/home/duval/DEV /mnt/nfs
- Automatiquement monté par /etc/fstab en ajoutant la ligne:

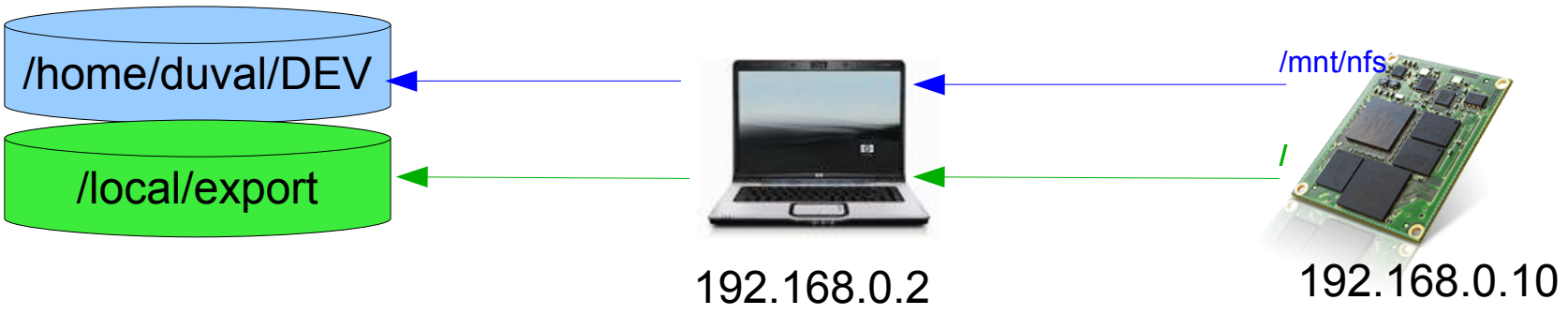
192.168.0.2:/local/export /mnt/host nfs hard,intr,rsize=8192,wsize=8192 0 0

Boot NFS (seulement le rootfs mais pas le noyau qui reste celui en flash):

Permet de tester les configurations du système

Copier le rootfs dans le répertoire de boot nfs de la station par exemple /local/export

BIOS>**setenv** rootpath /local/export
BIOS>**saveenv**
BIOS>**run** nfsboot



Écrire un driver Linux sur apf27

Définiton du driver:

Dans le répertoire armadeus/target/linux/modules

1- Créer le répertoire de ce driver
   mkdir armadeus/target/linux/modules/unDriver
2- Editer les références à ce nouveau driver dans les fichiers Makefile et Kconfig
3- Dans le répertoire unDriver:
   Mettre les codes source
   Créer les fichiers locaux Makefile et Kconfig (en s'inspirant d'autres modules)

Génération du driver:

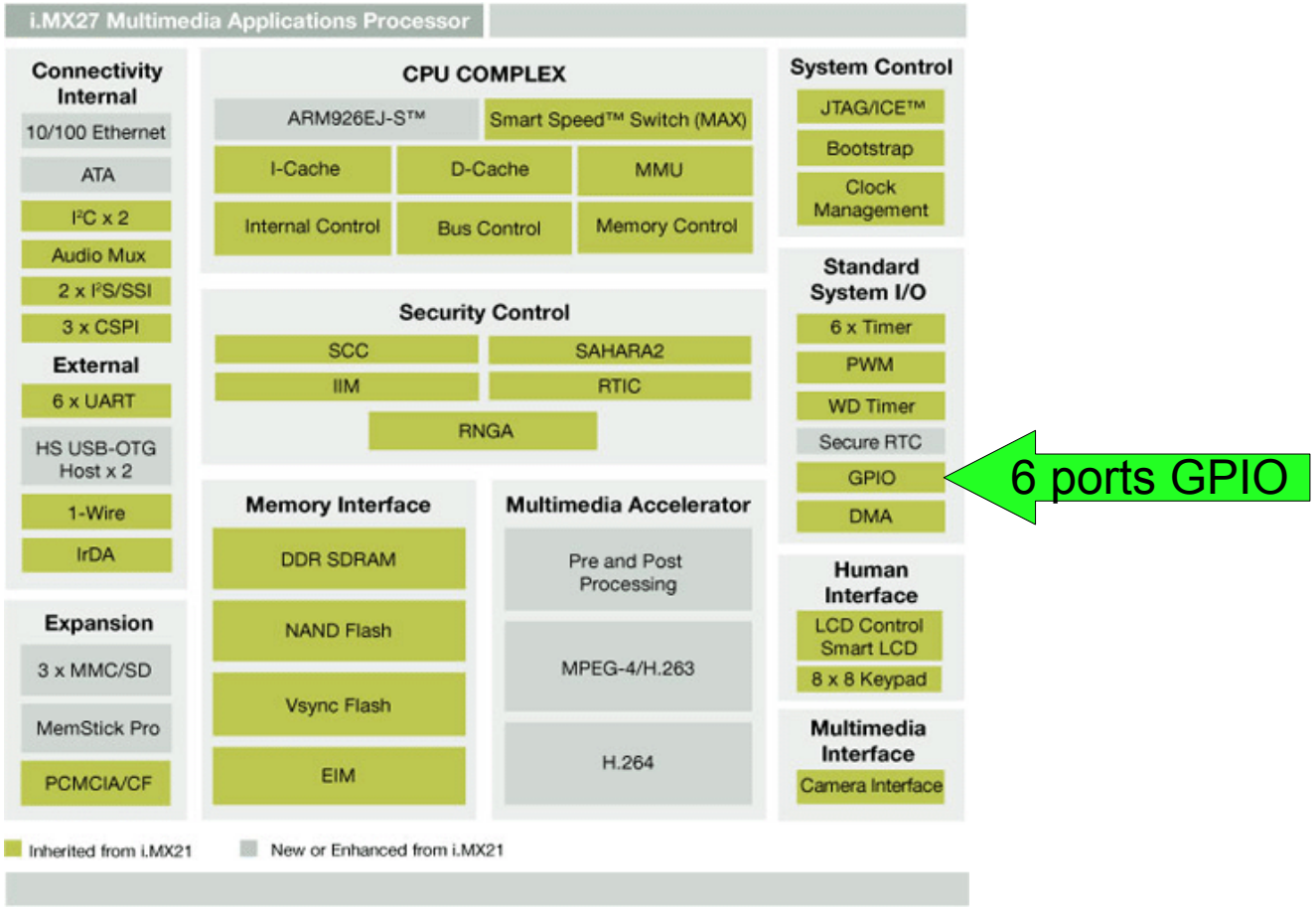
Pour le compiler
   make -C target/linux/modules/

Pour le compiler et créer le rootfs qui le contient
   make linux26

GPIO driver

Le processeur I.MX27 (freescale/multimedia) dispose de 6 ports 32bits general purpose IO (GPIO) qui peuvent:

- générer des IT sur front montant ou descendant
- être multiplexés entre diverses fonctions prédéfinies



GPIO driver standard

Le driver standard de GPIO permet d'accéder aux ports à travers le système de fichier via les fichiers de type /proc de type ascii utilisables dans des commandes du shell ou par programme (X=A-B-C-D-E-F).

1. /proc/drivers/gpio/portXmode pour configurer une pin comme gpio
2. /proc/drivers/gpio/portXdir pour lire par cat, ou parametrer par set, leur direction (in-0/out-1) pin par pin
3. /proc/drivers/gpio/portX pour lire par cat, ou parametrer par set, leur valeur/état pin par pin
4. /proc/drivers/gpio/portXirq: pour configurer les IT GPIO 0->pas d'IT, 1->up, 2->down, 3->up down
5. /proc/drivers/gpio/portXpullup: pour (de)activer pull-up interne du i.MX pour ce GPIO

Exemples:

Mettre en mode GPIO le bit 31 du port A

```
# echo -n 10000000000000000000000000000000 > /proc/driver/gpio/portAmode
```

Vérifier si les pin du port D sont en entrée ou sortie

```
 # cat /proc/driver/gpio/portDdir
 >>01101100111100000110110011110000       (ordre des pins: [31...0])
```

Mettre la pin 10 du port B en mode IT

```
# echo -n 00000000000000000000010000000000 > /proc/driver/gpio/portBirq
```

On peut aussi accéder bit par bit en lecture/écriture en utilisant directement les nœuds /dev associés aux fichiers-drivers /dev/gpio/PXy X = port (A-B-C-D-E) y= bit [0-31]

Exemple:

Lire le bit 0 du port D

```
# cat /dev/gpio/PD0
```

GPIO driver standard

Pour accéder par programme en C il existe des constantes IOCTL pour lire/écrire via **ioctl()** sur /dev ou on peut directement travailler avec des **read()/write()** sur les pseudos fichiers /proc

Ecriture formatée en ascii sur /proc

```
FILE *GPIO,*GPIODIR, *GPIOMODE;
char buffer[32];
char * bufferDir= "00000000011111111000000000000000";
char * buffer1=   "00000000000000011000000000000000";
char * buffer2=   "00000000000000110000000000000000";

   GPIODIR = fopen("/proc/driver/gpio/portDdir","w");
   setvbuf(GPIODIR,buffer,_IONBF,32);
   fwrite(bufferDir, sizeof(char), strlen(bufferDir), GPIODIR);
   fclose(GPIODIR);
   GPIO = fopen("/proc/driver/gpio/portD","w");
   setvbuf(GPIO,buffer,_IONBF,32);
   while(1) {
      fwrite(buffer1, sizeof(char), strlen(buffer1), GPIO);
      sleep(1);
      fwrite(buffer2, sizeof(char), strlen(buffer2), GPIO);
      sleep(1);
   }
   fclose(GPIO);
```

Ecriture via IOCTL sur /dev

```
int fd, i,  iomask,result;
unsigned char dev_buffer[BUF_SIZE+1];

  fd = open("/dev/gpio/portD", O_RDWR);
  printf("Opened on /dev/gpio/portD\n");
  iomask=0xFFFFFF00;
  ioctl(fd,GPIOWRDIRECTION,&iomask);
  iomask=0x003F0000;
  for (i=0;i<2;i++) {
    printf("Led ON\n");
    iomask=0x007F8000;
    ioctl(fd,GPIOWRDATA,&iomask);
    sleep(1);
    ioctl(fd,GPIORDDATA,&iomask);
    printf("read /dev/gpio/portD 0x%x\n",iomask);
    printf("Led OFF\n");
    iomask=0x00000000;
    ioctl(fd,GPIOWRDATA,&iomask);
    sleep(1);
  }
  close(fd);
```

GPIO driver modifié pour TP:usage

Vous utiliserez une variante de driver gpio, gpioFreq pour lire la fréquence du signal du Générateur en TP.

Il utilise l'**interruption** sur la pin 10 du port B pour débloquer la fonction **read()** qui n'est pas une vrai lecture mais un moyen de retourner la période qui a été mesurée depuis la dernière IT.

```
int main (i)
{
        int fd_input, retval = 0;
        unsigned long gpioWord;

        setItMask();

        fd_input = open ( "/dev/gpio/PB10", O_RDONLY);
        // set this process as owner of device file
        retval = fcntl (fd_input, F_SETOWN, getpid ());

        while (1) {
          // blocking read released on IT occurence
         read (fd_input, &gpioWord, sizeof(unsigned long));
         printf("USER: kernel passed period value %ld\n",
                                                    gpioWord);

          }
        }
        close (fd_input);  exit (EXIT_SUCCESS);
}
```

```
int setItMask()
{
  uint rsi;;
  char * bufferIrqMode="00000000000000000000010000000000";
  FILE *GPIOIRQ;
  char buffer[32*2];

  // Set the proper port in IT;
  GPIOIRQ = fopen( "/proc/driver/gpio/portBirq";,"w");
  setvbuf(GPIOIRQ,buffer,_IONBF,32*2);
  rsi = fwrite(bufferIrqMode, sizeof(char),
                        strlen(bufferIrqMode), GPIOIRQ);
  fclose(GPIOIRQ);
  return(0)
  }
```

Configuration en salle de TP

Serveur de fichiers locaux
http://192.168.0.1

Borne « ecole »

Borne CAES

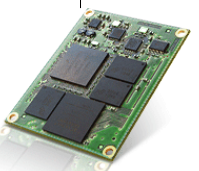
Lire ses mails

Récupérer des fichiers

192.168.0.1XY

Travailler en local

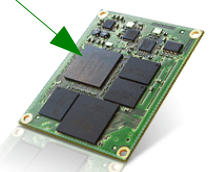
192.168.0.2



192.168.0.10

Travailler en remote

192.168.0.1XY



192.168.0.5X
Demandez une addresse

Configuration en salle de TP

Pour le WIFI installer WCID: sudo apt-get install wicd

Fichier /etc/interfaces

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 192.168.0.2
netmask 255.255.0.0
#gateway 192.168.0.1
```

Information

Linux embarqué et Linux en général:

http://free-electrons.com/docs/

La référence en matière de driver Linux:

Linux Device Drivers, Third Edition (O'Reilly)

Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman

Version libre en ligne http://lwn.net/Kernel/LDD3/