

# Conception d'un système Linux embarqué

Christophe BLAESS

[christophe.blaess@logilin.fr](mailto:christophe.blaess@logilin.fr)

<https://www.linkedin.com/in/christophe-blaess/>

<https://www.blaess.fr/christophe/>

twitter: @chrisblaess



Ingénierie et formations sur Linux et les logiciels libres

<https://www.logilin.fr>

<b>Environnement Linux embarqué.....</b>	<b>3</b>
Spécificités de l'environnement embarqué.....	4
Développement applicatif embarqué, <i>cross-compilation</i> .....	5
Composants d'un système embarqué.....	6
<i>Build system</i> : Buildroot.....	7
<i>Build system</i> : Yocto.....	8
Travaux pratiques : Création d'un système embarqué avec Buildroot.....	9
Pour information : utilisation de Yocto.....	10
<b>Composition d'un système Linux embarqué.....</b>	<b>11</b>
Aspects matériels.....	11
Spécificités d'un système Linux embarqué.....	15
<b>Boot du système Linux.....</b>	<b>16</b>
Bootloader et kernel.....	16
Travaux pratiques : Lancement de l'émulateur.....	17
Carte SD d'un Raspberry Pi.....	19

Ce support de formation est distribué sous licence **Creative Commons 4.0**



*(Attribution - Partage dans les mêmes conditions).*

Vous êtes libres de copier et partager ce document, en mentionnant son origine. Si vous l'intégrez dans un contenu plus vaste, ce dernier devra être distribué avec les mêmes droits.

Ce cours a été rédigé en utilisant des logiciels libres sur système d'exploitation Linux :

- *LibreOffice Writer* pour le support et la mise en page
- *LibreOffice Draw* pour les dessins vectoriels
- *Gimp* pour les images bitmap

Linux embarqué

ILE v. 5.4

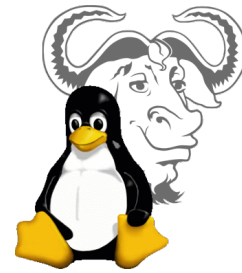
<https://www.blaess.fr/christophe/>

<https://www.logilin.fr>

# Environnement Linux embarqué

## Linux

- Noyau du système d'exploitation GNU/Linux compatible Unix,
- environnement constitué de logiciels libres,
- développement essentiellement bénévole.



*Linux n'a jamais été prévu pour être un système industriel, temps-réel ou embarqué !*

## Système embarqué (*embedded system*)

« **to embed** (*embedded, embedding*) : (v. transitif) : enfoncer, sceller, incruster. »

(*Dictionnaire bilingue Larousse*).

- Système informatique souvent non-perçu comme tel,
- autonome, robuste, résilient,
- intégré comme élément d'un système plus large.

Source logo « Gnu/Linux » : <https://commons.wikimedia.org/wiki/File:Gnulinux.png>  
Manchot « Tux » créé par Larry Ewing, mascotte « GNU Head » créée par Aurelio A. Eckert.

Influences de Linux :

1969 – Ken Thompson & Denis Ritchie : Unix AT&T

1978 – Université de Berkeley : Unix BSD

1984 – Richard Stallman : Projet Gnu ([www.gnu.org](http://www.gnu.org))

1987 – Andrew Tanenbaum : Minix

Création de Linux :

1991 – Linus Torvalds : Linux 0.0.1

## Voir aussi

- [TANENBAUM] : Andrew Tanenbaum & Albert Woodhull – *Operating Systems, Design and Implementation*.
- [RAYMOND] : Eric S. Raymond – *The Art of Unix Programming*.
- [TORVALDS] : Linus Torvalds & David Diamond – *Just for Fun : The Story of an Accidental Revolutionary*

## Spécificités de l'environnement embarqué

### ***Contraintes matérielles***

Performance : puissance CPU, capacité mémoire, stockage, richesses I/O...

Encombrement : ventilateur / dissipateur, batterie, connecteurs...

Autonomie : batterie, panneau solaire, alimentation externe...

### ***Contraintes logicielles***

Performance : optimiser les ressources matérielles, temps réel...

Robustesse : fonctionnement 7/7 & 24/24, environnement hostile...

Sécurité : détection d'intrusion ou de modification, mises à jour, signatures...

### ***Contraintes économiques***

Concurrence : qualité, IHM, ergonomie...

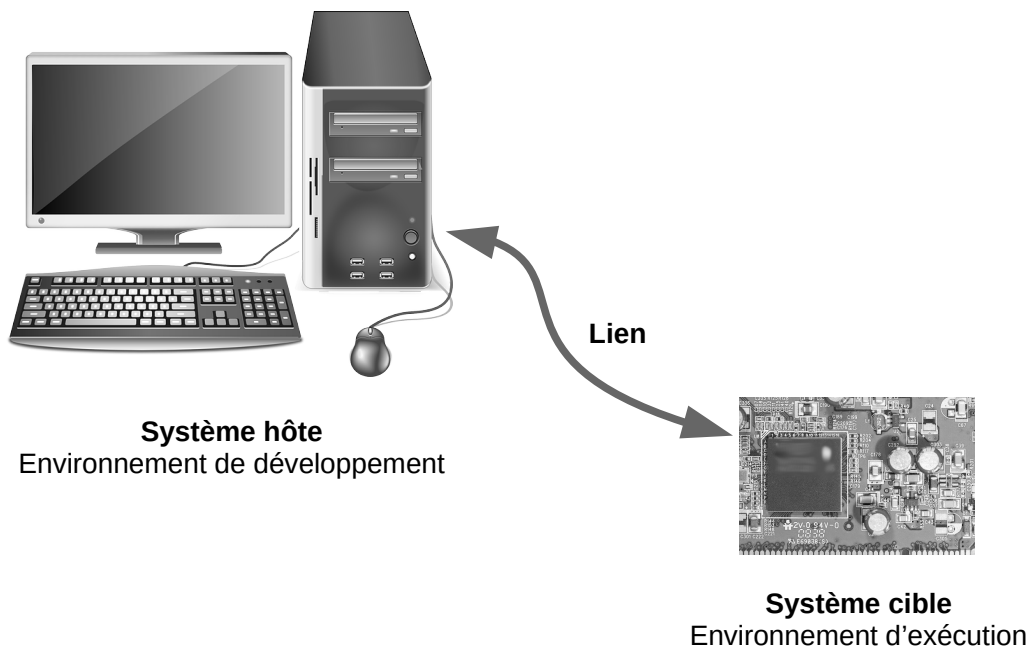
Coûts : choix des composants, licences logicielles...

Présence d'un « *market* » applicatif ouvert...

## Voir aussi

- [FICHEUX] : Pierre Ficheux – *Linux embarqué : Mise en place et développement*.
- [BLANC] : Gilles Blanc – *Linux embarqué : Comprendre, développer, réussir*.

## Développement applicatif embarqué, *cross-compilation*



Sources graphiques : images 158675 et 3262915 sur Pixabay.

### Système hôte :

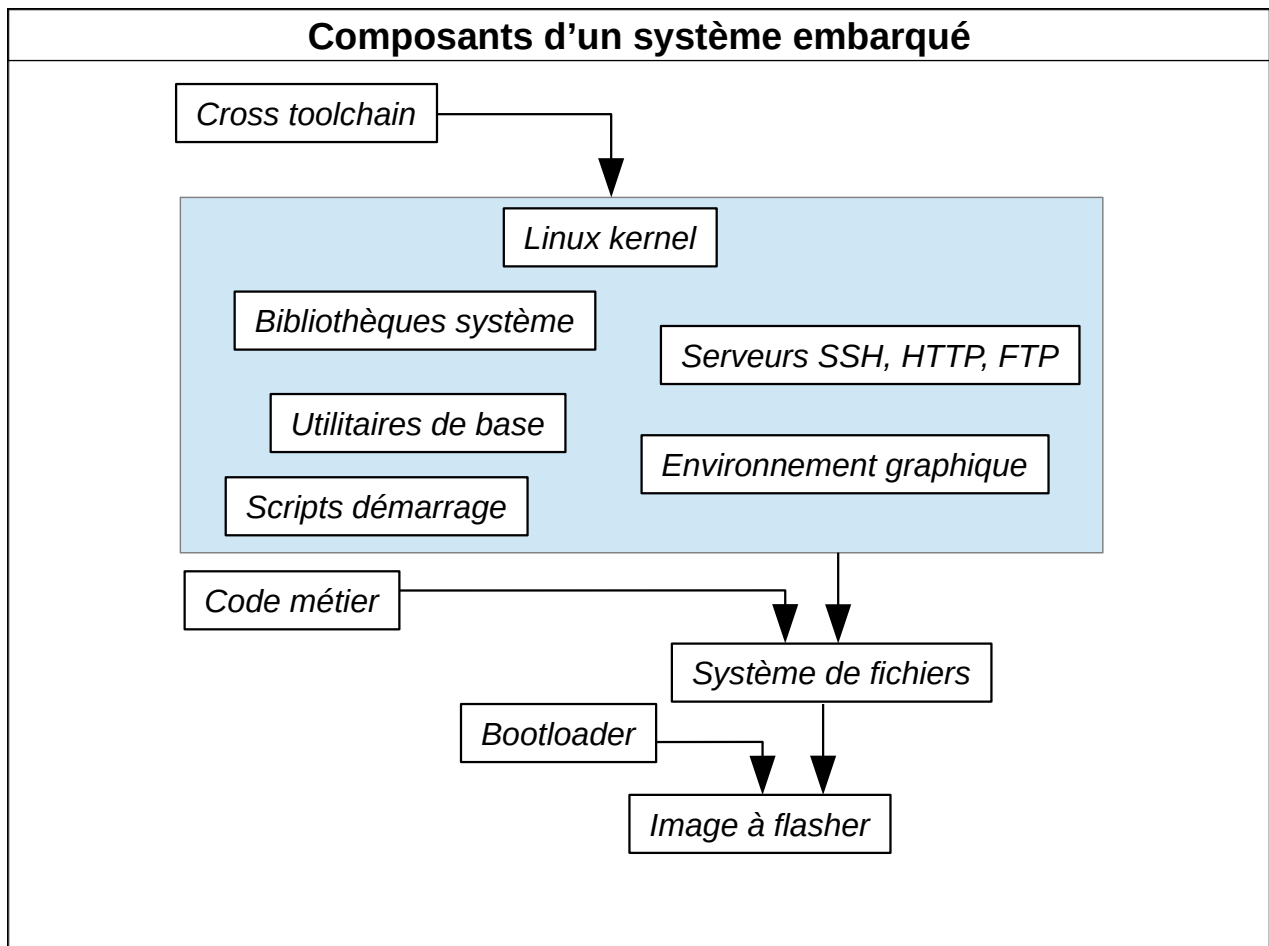
- Linux (éventuellement dans une machine virtuelle) : une distribution classique intégrant Eclipse et une chaîne de cross-compilation Gnu ;
- Windows : avec les outils Gnu provenant des projets Cygwin ou Mingw ;
- Autres Unix : outils Gnu standards.

### Système cible :

- Linux sur processeurs x86 (32/64 bits), PowerPC, Motorola 68000, Arm, SuperH, Mips ;
- systèmes non-Linux : microcontrôleurs, RTEMS, FreeRTOS...

### Lien :

- Série RS/232 : terminal texte, transfert de fichiers par *Xmodem* ;
- Ethernet et pile IP : *telnet* ou *ssh*, *ftp*, *scp*, montage par *NFS* ;
- Interface JTag : flashage direct et débogage au niveau du processeur...



La création manuelle des différents composants d'un système embarqué est de plus en plus rare (développement très spécifique et minimaliste, modification en profondeur des composants de base, aspect pédagogique, etc.)

On dispose aujourd'hui de plusieurs outils de génération de plates-formes Linux embarqué.

### Voir aussi

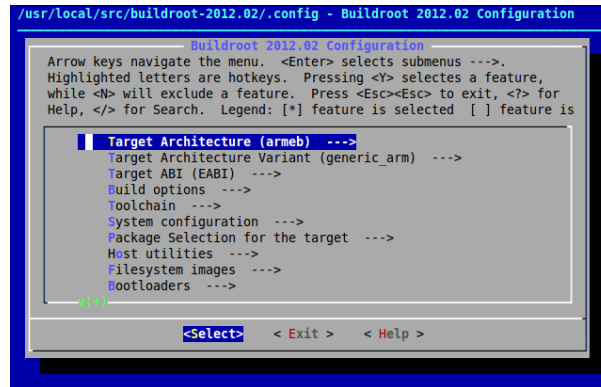
[BLAESS] Christophe Blaess – *Raspberry Pi from Scratch* – GNU/Linux Magazine France 155 & 158 – Diamond Editions – Disponible sur <https://www.blaess.fr/christophe/articles/files-glmf/>.

[FICHEUX] Pierre Ficheux – « *Distributions embarquées pour Raspberry Pi* » – Open Silicium n° 7 – Diamond Editions.

## **Build system : Buildroot**

**Buildroot** est un ensemble de *Makefiles*, de scripts et de fichiers de configuration permettant la construction complète d'un système Linux pour une cible embarquée.

Il télécharge automatiquement les paquetages nécessaires pour la compilation et l'installation.



La configuration de Buildroot est réalisée à travers une interface assez simple d'utilisation.

Buildroot s'appuie intensivement sur l'utilitaire make comme moteur de compilation.

Buildroot permet de construire une image complète « prête à flasher » comprenant tout l'environnement d'exécution (noyau, bibliothèques, utilitaires, applications, système graphique, etc.).

### **Avantages :**

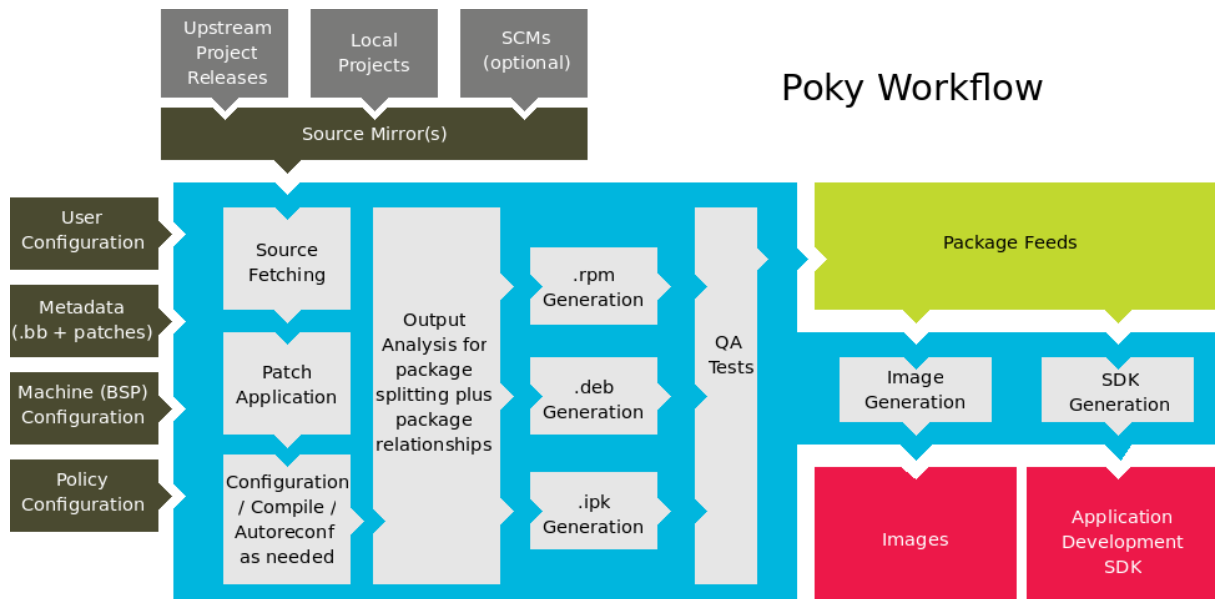
- Système relativement simple à comprendre et à modifier
- Possibilité d'écrire facilement des recettes pour ajouter des composants système.
- Intégration aisée du code métier.

### **Inconvénients :**

- Pas de gestion de packages, génération d'une image contenant tout le système.
- Difficile de gérer des versions différentes (gamme de produits) dans la même arborescence.

## Build system : Yocto

Outil complet pour créer une plate-forme embarquée et son environnement de développement d'applications-métier.



Source : <http://www.yoctoproject.org>

### Avantages :

- Très complet et très riche.
- Système de *packages* avec dépendances dans le code généré pour la cible.
- Possibilités de configuration et d'adaptation très complètes.

### Inconvénients :

- Système assez complexe à maîtriser.
- Durée de compilation initiale longue en raison du nombre importants de packages inclus.



## Travaux pratiques : Création d'un système embarqué avec Buildroot

*Téléchargez une version de Buildroot*

```
$ wget https://www.buildroot.org/downloads/buildroot-2024.02.6.tar.xz
$ tar xf buildroot-2024.02.6.tar.xz
$ cd buildroot-2024.02.6
```

*Préparez une configuration pour émulateur Arm (« O majuscule »)*

```
$ make O=../build-qemuarm qemu_arm_vexpress_defconfig
$ cd ../build-qemuarm
```

*Vérifiez la configuration :*

```
$ make menuconfig
```

*Dans le menu « Toolchain », modifiez*

Toolchain type (External toolchain)

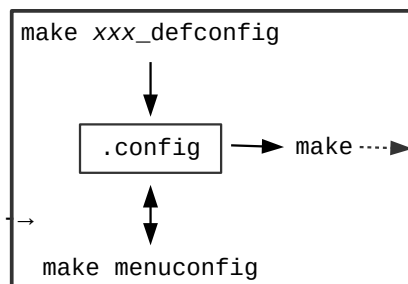
[\*] Copy gdb server to the Target

*Dans le menu « Build Option », modifiez*

Download dir : **\$(TOPDIR)/../dl/**

*Lancez la compilation (environ une heure) :*

```
$ make
```



L'utilisation de fichier « .config » est une convention que l'on retrouve dans plusieurs projets libres (Linux, Buildroot, Busybox...).

Si on exécute Buildroot sous l'identité `root`, une erreur peut se produire. Pour l'éviter il faut saisir, avant de lancer `make` la ligne : **`export FORCE_UNSAFE_CONFIGURE=1`**

Au lieu de `make menuconfig`, vous pouvez utiliser : `make xconfig` ou `make gconfig` mais ces configurations ne fonctionnent que si les bibliothèques graphiques correspondantes (Qt et Gtk) sont installées.

Une bonne pratique consiste à utiliser l'option **`O=`** de buildroot pour éviter de polluer les sources de Buildroot.

### Voir aussi

Article en ligne : « Créer un système complet avec Buildroot »

<https://www.blaess.fr/christophe/articles/creer-un-systeme-complet-avec-buildroot/>

## Pour information : utilisation de Yocto

*Récupérer les sources de Yocto (quelques minutes) :*  
**\$ git clone git://git.yoctoproject.org/poky.git -b kirkstone**

*Préparer la configuration de Yocto*  
**\$ source poky/oe-init-build-env build-qemuarm**

*Éditer conf/local.conf*  
**MACHINE = "qemuarm"**

*Lancer la compilation (environ quatre heures)*  
**\$ bitbake core-image-base**

*Vérifier le résultat de la compilation :*  
**\$ cd tmp/deploy/images/qemuarm**  
**\$ ls**  
**[...] core-image-base-qemuarm.ext4 [...]**

*Lancer l'émulation*  
**\$ runqemu qemuarm**

### Voir aussi

Article en ligne : « Linux embarqué avec Yocto Project »

<https://www.blaess.fr/christophe/yocto-lab/index.html>

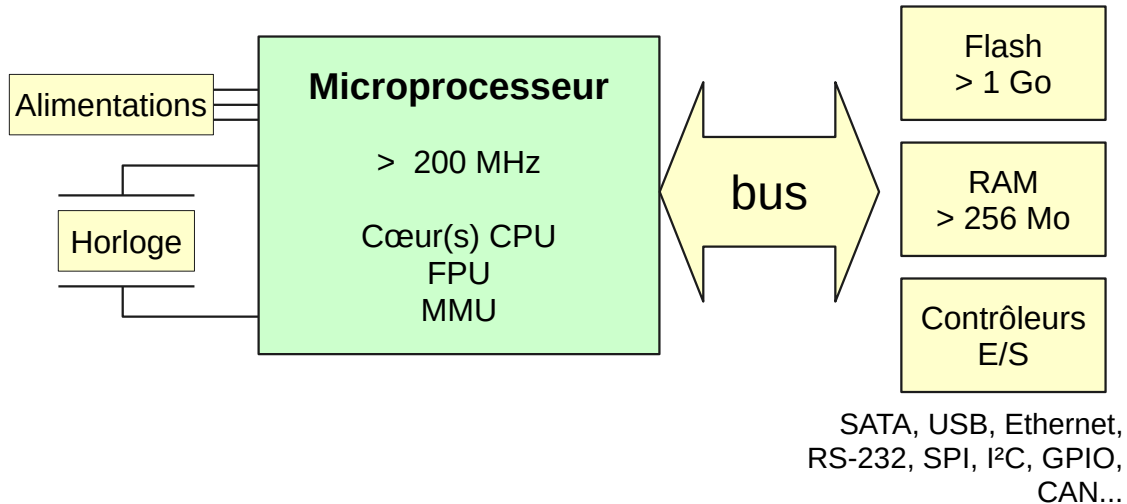
[SALVADOR] : Otavio Salvador, Daiane Angolini – *Embedded Linux Development with Yocto Project*.

[TEXIER] : Pierre-Jean Texier, Petter Mabäcker - *Yocto for Raspberry Pi*

# Composition d'un système Linux embarqué

## Aspects matériels

### Microprocesseur



Entrées-sorties réalisées par des contrôleurs externes au processeur.

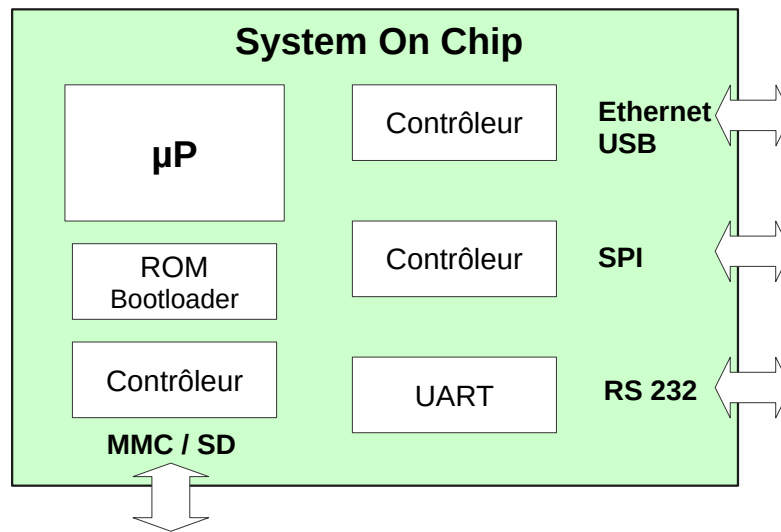
Mise en œuvre directe électroniquement très complexe.

Optimisé pour l'utilisation d'un système d'exploitation.

Quelques exemples de microprocesseurs :

- Famille Arm : ARMv7 (Cortex-A8, Cortex-A9, Cortex-A15, Cortex-M...) ARMv8 (Cortex-A35, Cortex-A53, Cyclone, Exynos, Cortex-A76, ...)
- Famille x86 : Intel (Atom, Core i5, i7...), AMD (Opteron, Phenom...), Via (C3, C7, Nano...)
- Famille M68k : Motorola 680x0, Coldfire (MCF5xxx), Dragonball.
- Famille PowerPC : Apple (G5), IBM (Power 6, Power 7, Power 8, Cell, Xenon)

### System on Chip (S.O.C.)



Contrôleurs d'entrées-sorties déjà incorporés.

Intégration électronique encore assez complexe.

Entrées-sorties industrielles (CAN) ou analogiques (ADC/DAC, PWM) assez rares.

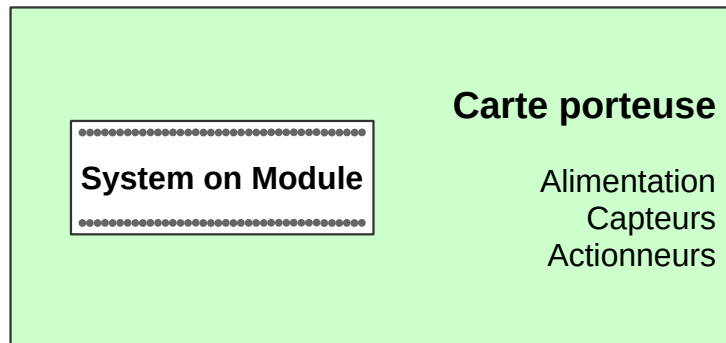
Quelques *systems-on-chip* Arm :

- Allwinner : A13, A20, A63, A80...
- Broadcom : BCM2835, BCM2837...
- NXP (Freescale) : LPC, i.MX21, i.MX23, i.MX6, i.MX8, OorIQ...
- Marvell : 88SE6, 88SE9...
- Rockchip : RK30xx, RK31xx, RK32xx, RK33xx...
- Texas Instruments : OMAP, DaVinci...

## System-on-module (SOM)

Un **module** regroupe un *system-on-chip* et quelques composants (mémoire RAM et flash par exemple).

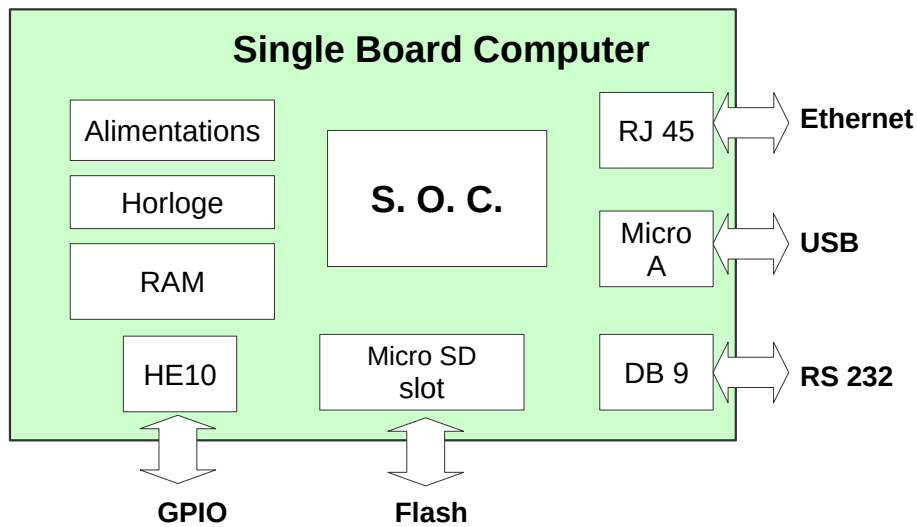
On le connecte à une carte porteuse contenant l'alimentation et l'électronique de communication externe.



Le développement de la carte porteuse est facilement réalisable par un bureau d'étude électronique.

Exemples de fournisseurs de modules : Phytex (phyCore, phyFlex...), Armadeus Systems (OposSOM, APF...), Marvell (Armada...), Acme Systems (Aria, Arietta, Acqua...), Raspberry Pi Compute Module etc.

### *Single-Board-Computer (SBC)*



Ordinateur **mono-carte** intégrant *system-on-chip*, mémoire, connecteurs d'E/S, etc.  
Très utilisé pour le prototypage et la mise au point initiale.

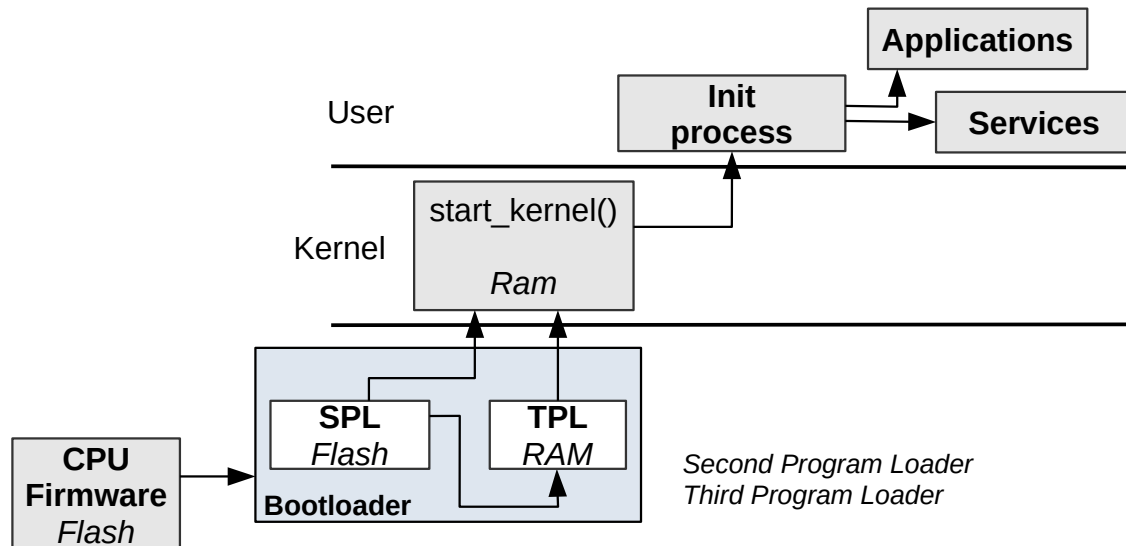
Exemples : BeagleBone Black, Raspberry Pi, Banana Pi, OLinuXino...

Certains SBC (sans Linux) reposent sur des microcontrôleurs : Arduino, Launchpad, Teensy...

Spécificités d'un système Linux embarqué		
Applications finales	LibreOffice, Firefox, Gimp, VLC, Eclipse, etc.	Logiciels spécifiques, code métier.
Environnement graphique	X-Window (serveur Xorg). Environnements Gnome, Kde, etc.	DirectFB. LibEFL, QtEmbedded, GTK+
Outils de développement	Chaîne de compilation GNU native, outils de mise au point.	Chaîne de cross-compilation sur l'hôte et débogage distant.
Shell	Bash, Ksh, Zsh, Dash, etc.	Ash dans Busybox
Utilitaires système	GNU Coreutils, Net-tools, Procps. Open SSH, Apache...	Busybox. Dropbear, Lighttpd...
Initialisation Boot	Systemd, Init system V NetworkManager, Kmod...	Scripts pour Busybox
Bibliothèques	Standards. Gnu Glibc...	Réduites et adaptées, uClibc, musl...
Noyau	Universel, fourni par la distribution, nombreux modules.	Ajusté précisément. Uniquement les modules nécessaires.
Matériel	<b>Poste de travail. Serveurs.</b>	<b>Systèmes embarqués.</b>

# Boot du système Linux

## Bootloader et kernel



Il existe plusieurs *bootloaders* par exemple : Uboot, Barebox, Grub, etc.

1. Une mémoire flash, EEPROM, ou ROM contient un **firmware** (micro-code) immuable qui initialise le CPU, et transmet le contrôle au *bootloader*. Ce micro-code contient parfois une interface minimale pour paramétrer le chargement du *bootloader*.
2. Le **bootloader** (chargeur de démarrage) prend le relais, pour charger une image du noyau Linux (placée en mémoire flash, sur une partition de carte SD, depuis un réseau, etc.).
3. Le **noyau Linux** s'initialise, détecte les périphériques et monte son système de fichiers principale. Il cherche un fichier `init` qu'il exécute.
4. Le processus **init** configure le système depuis l'espace utilisateur et démarre les applications-métier.



## Travaux pratiques : Lancement de l'émulateur

*Vérifiez le résultat de la compilation :*

```
$ ls images/  
rootfs.ext2  start-qemu.sh  vexpress-v2p-ca9.dtb  zImage
```

*Lancez l'émulateur avec les paramètres suivants :*

```
$ qemu-system-arm \
  -M vexpress-a9 \
  -smp 1 \
  -m 256 \
  -kernel images/zImage \
  -dtb images/vexpress-v2p-ca9.dtb \
  -drive file=images/rootfs.ext2,if=sd,format=raw \
  -append "console=ttyAMA0,115200 rootwait root=/dev/mmcblk0" \
  -serial stdio \
  -net nic,model=lan9118 \
  -net user \
```

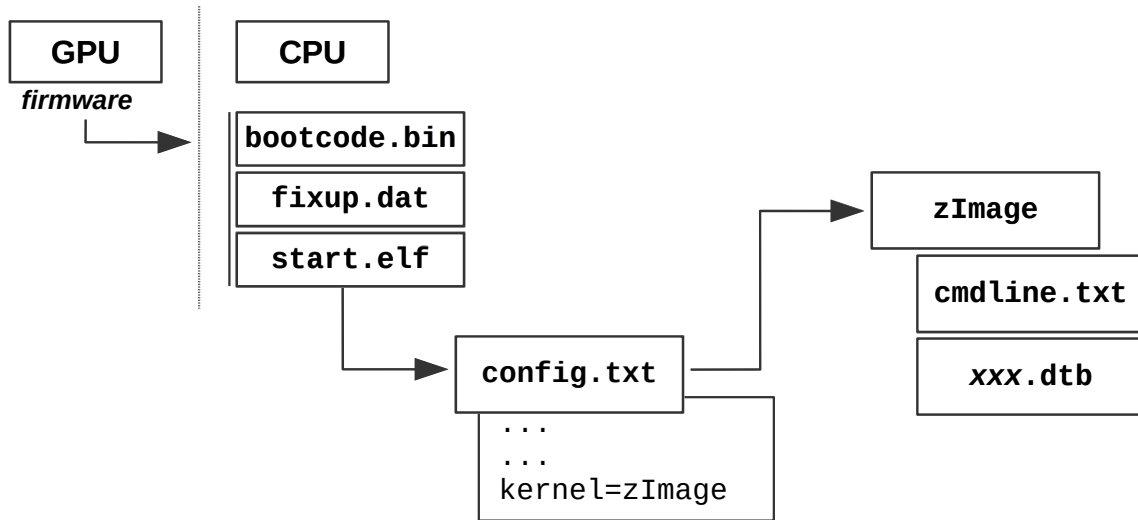
*ou utilisez le script fourni*

```
$ images/start-qemu.sh
```

*Pour pouvoir utiliser la séquence CTRL-C dans l'émulateur sans le tuer,  
programmez au préalable une autre séquence de touches d'interruption :*

```
$ stty intr ^t
```

### Cas particulier : boot du Raspberry Pi



La GPU du Raspberry Pi charge un *bootloader* propriétaire (fourni par Broadcom) en mémoire, pour qu'il soit exécuté par le CPU.

Ce *bootloader* s'appelle **bootcode.bin** et doit se trouver sur la première partition de la carte micro-SD, formatée avec le système de fichiers **vfat** (fat 32).

Le *bootloader* lit un fichier de configuration nommé **config.txt** se trouvant sur la même partition.

Le *bootloader* utilise ensuite deux fichiers : **fixup.dat** et **start.elf** qui chargeront à leur tour le noyau dont le nom est précisé dans **config.txt** (ici **zImage**). En outre le contenu du fichier **cmdline.txt** est passé en paramètre au noyau ainsi que la configuration matérielle décrite dans le *device tree* (fichier **.dtb**)

## Carte SD d'un Raspberry Pi

