Introduction à l'embarqué et à OpenWRT – 10/01/2011, pour le master2 I2L

Sébastien Bocahu <zecrazytux@zecrazytux.net> –

Embarqué ?

Définir rapidement ce qu'est l'embarqué, quelles sont les principales différences entre la plateforme x86 que nous connaissons bien et les plateformes embarquées.

Étapes typiques pour la mise en place de GNU/Linux sur plateforme embarquée

Des logiciels libres et GNU/linux pour l'embarqué

Présentation d'outils libres dédiés ou utilisables pour l'embarqué. Bootloader, création de chaine de crosscompilation, création de rootfs, buildsystems, distributions

Le projet OpenWRT

OpenWRT est une distribution GNU/Linux pour plateforme embarquée, historiquement concentré sur les routeurs et point d'accès wifi.

Récemment, OpenWRT s'étend avec un support pour X11, GTK+/Qt.

> *Un système embarqué peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise. Ses ressources disponibles sont généralement limitées. Cette limitation est généralement d'ordre spatial (taille limitée) et énergétique (consommation restreinte). (**Wikipedia**)*

Notre système d'exploitation doit convenir à ces limitations. On retrouve généralement ces contraintes:

- Architecture CPU, voire bus spécifiques
- Faible espace de stockage
- Faibles performances

principalement dans le but de réduire les coûts et la consommation énergétique

Auquelles s'ajoutent d'autres éventuelles contraintes selon le produit:

- Ordonnancement temporel des tâches, temps réel
- Éxigence de sécurité, de confidentialité
- Éxigence de non-défaillance (pensez au matériel à bord d'un avion)

Électronique grand publique

- “Box” ADSL, routeurs, NAS
- Télévisions, lecteurs DVD/multimédia
- appareils photos, caméras numériques
- téléphones portables, smartphones

Industrie

- Alarmes, télésurveillance
- Automates
- Aviation, satellites

- Architecture processeur différente
  - ARM
  - MIPS
  - PowerPC
  - parfois x86…
- Pas de “BIOS”, initialisation du matériel par le bootloader, par exemple uboot
- Pas de disque dur / stockage amovible: stockage du système d'exploitation en Flash, souvent directement sur la carte “mère”
  - NOR / NAND
  - quelques Mo à quelques centaines de Mo
- Quelques Mo à quelques centaines de Mo de RAM
- Bus spécifiques: SPI, CAN, I2C…
- Pas d'écran et de clavier ! pour développer/débugguer/installer:
  - Serial (souvent TTL 3,3v)
  - JTAG (debug/recovery)
- utilisation de carte de développement pour prototyper

Bien que n'ayant pas été conçu pour l'embarqué, le couple GNU/Linux, et de nombreux projets libres qui gravitent autour s'adaptent aux limitations des plateformes embarquées.

Les avantages sont nombreux: (avantages “classiques” du logiciel libre)

- Bon support matériel: le kernel Linux supporte de nombreuses architectures, dont ARM, MIPS, POWERPC…
  - contributions parfois directes de constructeurs
- réutilisation de librairies, projets éprouvés
- modifications, personnalisations aisées (modulaire)
- faible coût
- documentation, support de la communautée
- Licences libres (reste le respect de celles-ci…)

Crosscompilation

Compilation d'un logiciel pour une architecture cible différente de l'architecture actuelle (hôte)

Utilisation d'une chaine de compilation croisée (crosscompilation toolchain) qui contient:

- Le compilateur, linker, etc (Gcc par exemple)
- Les en-têtes (headers, .h) de la libc, du kernel et d'éventuelles autres librairies
- Les librairies nécessaires compilées pour l'architecture cible

Avantages de la crosscompilation:

- Rapidité de compilation (machine de build performante vs embarqué peu performant)
- Stockage disponible
- évite la mise en place d'un environnement sur la plateforme embarquée (minimalisme), il est plus simple d'installer ces outils sur une distribution complète sur une workstation

Configuration du bootloader: Uboot

> *Das U-Boot (Universal Bootloader) est un chargeur de démarrage pour différentes architectures, telles que PPC, ARM, AVR32, MIPS, x86, 68k, Nios, et MicroBlaze.*

- Supporte de nombreux systèmes de fichier, dont certains dédiés au stockage “Flash”: jffs2, yaffs2, ubifs (fat, ext2, ext3…)
- Supporte de nombreux OS/kernels (dont Linux)

- Permet de charger kernel/rootfs via tftp, nfs, serial

```
uboot> printenv bootargs
bootargs=console=ttyS0,115200 ubi.mtd=2 root=ubi0:rootfs rootfstype=ubifs'

uboot> tftpboot 0x6400000 uImage-2.6.30-sheevaplug
uboot> bootm 0x6400000
```

Cours/Présentation sur Uboot

Compilation du kernel Linux

Il nous faut un kernel !

- Quelles sources ?
  - Y a t'il un support mainstream suffisant ?
  - Nombreuses branches de développement, branches de contributeurs
  - Besoins de patchs ?
- Quelle configuration ?
  - Matériel spécifique, qui ne changera pas
  - Besoin d'optimisations
  - Exit les kernel GENERIC !

```
$ make ARCH=arm ${MACHINE}_defconfig
```

- Initrd ou non ?

Compilation du kernel Linux

Crosscompilation du kernel

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
```

Création d'une image bootable par uboot

Uboot peut charger des images au format ELF (Executable and Linkable Format), ou son propre format: Uboot Image format, qui ajoute des informations sur le type du système d'exploitation, le point d'entrée… et est souvent préféré.

```
mkimage -A arm -O linux -T kernel -C none -a 30008000 -e 30008000 -n "Kernel" -d linux.bin uImage
```

Création du système de fichier racine (rootfs)

Par exemple, debootstrap avec le repository Emdebian/grip, en compilant busybox, ou en utilisant un outil/framework de création de rootfs (nous verrons OpenWRT par la suite).

```
$ sudo debootstrap lenny grip/ http://www.emdebian.org/grip/
$ cd grip/dev; /dev/MAKEDEV -v generic console
... boot init=/bin/sh
/debootstrap/debootstrap --second-stage
```

Nous évoquerons par la suite quelques outils pour la création de rootfs

Ajouts d'éventuels logiciels tiers (packaging ?)

Crosscompilation “a la main”

Utilisation d'une toolchain précedement installée/compilée

- Appel du compilateur:

```
$ arm-linux-gnueabi-gcc -o hello hello.c
```

- Autotools/autoconf:

```
$ ./configure --host=arm-linux-gnueabi
$ make
```

- Makefile sans autoconf:

```
$ make CC=arm-linux-gnueabi-gcc
```

Crosscompilation “a la main”

Variables d'environement de pkgconfig (souvent utilisé dans les Makefile)

```
$ export PKG_CONFIG_LIBDIR="/usr/arm-linux-gnueabi/usr/lib/pkgconfig"
$ export PKG_CONFIG_LIBDIR="$PKG_CONFIG_LIBDIR:/usr/arm-linux-gnueabi/usr/share/pkgconfig"
$ export PKG_CONFIG_SYSROOT_DIR="/usr/arm-linux-gnueabi"
```

Problèmes au moment de l'édition de liens

Si **ld** tente d'utiliser les librairies de l'hôte, ça ne fonctionnera evidemment pas…

Indiquez à **ld** où se trouvent les librairies:

```
LDFLAGS="-L/usr/arm-linux-gnueabi/lib -L/usr/arm-linux-gnueabi/usr/lib"
LDFLAGS="$LDFLAGS -Wl,-rpath-link /usr/arm-linux-gnueabi/lib"
LDFLAGS="$LDFLAGS -Wl,-rpath-link /usr/arm-linux-gnueabi/usr/lib"
```

-L: emplacement des librairies
-rpath-link: emplacement des librairies, nécessaire dans certains cas lors d'utilisation d'ELF ou sur Sun OS (voir la page de man)

Problèmes à l'utilisation de libtool

-inst-prefix-dir permet d'installer les binaires dans un *DESTDIR* différent de l'emplacement réservé au logiciel dans le futur (stagging area). **Libtool** assure également que les binaires sont bien liés avec les librairies présentes dans ce répetoire. Il nous faut donc utiliser cet argument, avec comme paramètre l'emplacement de notre toolchain.

Problème: il peut être nécessaire de patcher libtool pour la prise en compte de notre répertoire pour que l'édition de lien se passe correctement !

```
~2784 ltmain.sh
+ test -f "$inst_prefix_dir$add" && add="$inst_prefix_dir$add"
```

Nous évoquerons par la suite quelques outils pour la création de packages

libC

- Glibc: GNU libc, complète, peu adaptée
- Eglibc: Embedded GNU libc, GNU libc patchée, modulaire
- uClibc: libc des plus utilisée en embarquée: très petite
- dietlib
- newlib
- BSD libc

Userland “de base”

- outils classiques, “complets”: coreutils, outils GNU
- Busybox: le couteau suisse pour l'embarqué ! regroupe plusieurs centaines de commandes en un exécutable de quelques centaines de Ko
- BSD: beastiebox, reprend le concept de busybox avec l'userland BSD

bootloaders

- Uboot
- Qi
- Redboot

Création de chaine de compilation croisée (cross toolchain)

- Buildroot (uClibc)
- Crossdev (Gentoo)
- Crosstool / Crosstool-ng
- Bitbake (OpenEmbedded)
- Crossdev/tsrpm (Timesys), OSELAS.Toolchain()
- (Emdebian emchain)

Buildsystem, création de rootfs, compilation de logiciels, création de packages

- Buildroot (uClibc)
- Scratchbox (Maemo)
- Bitbake (OpenEmbedded)
- PTXdist
- LTIB (Linux Target Image Builder)
- CLFS ?
- Certaines distributions ont un support de crosscompilation: Gentoo, Emdebian, rpmbuild pour les distribution basées sur RPM… ou création de rootfs: debootstrap, cdebootstrap, emsandbox, urpmi…

Qemu

> *Qemu est un logiciel d'émulation qui supporte de nombreuses architectures, dont ARM, MIPS, PPC… C'est un excellent outil pour tester, mais aussi debugguer grâce à son excelente intégration avec gdb*

User mode emulation:

```
$ qemu-arm -L /usr/arm-linux-gnueabi ~/work/bin/mysoft
```

System emulation:

```
$ qemu-system-arm -kernel vmlinuz-arm -hda rootfs.img -append "root=/dev/sda1"
```

Debugging:

```
arm$ gdbserver 0.0.0.0:9999 mysoft
work$ arm-linux-gnueabi-gdb target/bin/mysoft
(gdb) target remote 127.0.0.1:9999
```

- Debian: support de plateformes ARM, MIPS… mais n'est pas dédiée à l'embarqué (lourd…)
- Emdebian: Debian pour l'embarqué !
- OpenEmbedded: Angstrom, Poky, SlugOS, SHR… (ou encore feu Openmoko, OpenZaurus)
- **OpenWRT**
- Distributions commerciales (Montavista…)
- CLFS ?
- …

OpenWRT est une distribution Busybox/Linux minimaliste. Elle se concentre surtout sur le support de matériels réseaux: routeurs, points d'accès wifi…

OpenWRT est également un set d'outils qui permettent de construire une rootfs / firmware personalisé.

OpenWRT intègre un gestionnaire de packages et met à disposition des mirroirs de paquets.

De nombreux matériels grand public sont supportés par OpenWRT.

OpenWRT intègre tout un buildsystem (dérivé de buildroot) qui permet de:

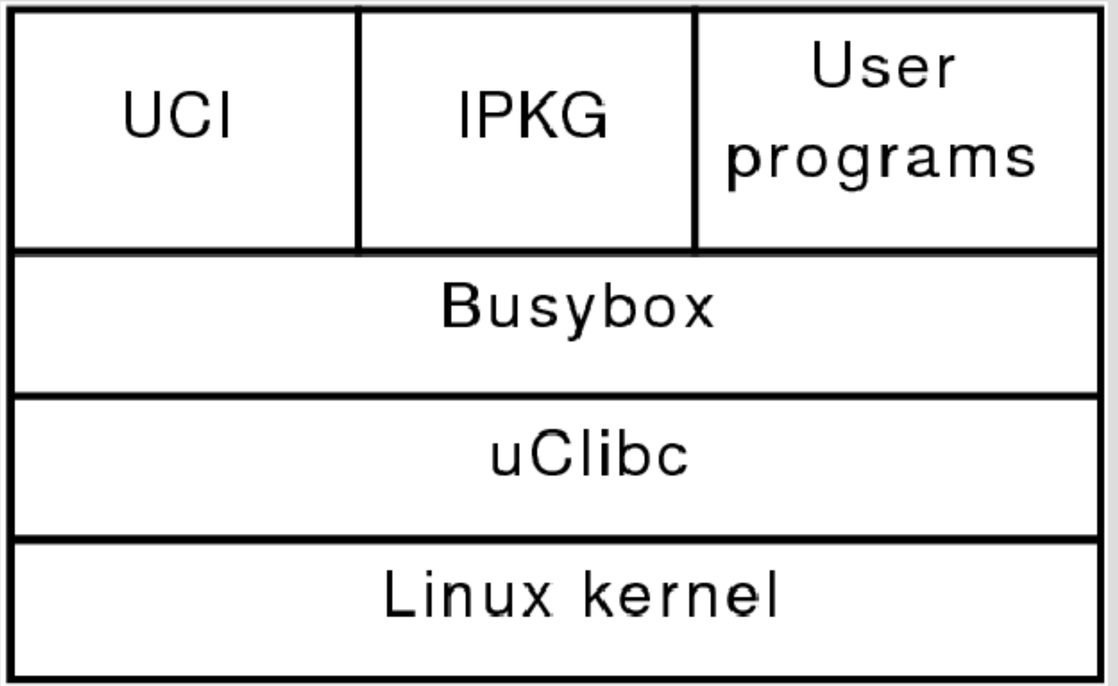
- Générer une chaine de compilation croisée, en utilisant la librairie uclibc
- Compiler le kernel Linux avec les options et les patchs adéquats
- Générer une rootfs personalisée (busybox, userland de base, configuration réseau…)
- Générer des paquets IPKG, qui pourront être distribués sur un mirroir et installés par le package manager opkg

Toutes ces fonctionnalités sont fournies par des Makefiles et des scripts, avec un support dédié à certaines architectures (Mips, ARM particulièrement) et machines (Kirkwood par exemple, est une plateforme ARM présente dans nos Guruplugs).

OpenWRT utilise un système de configuration spécifique: UCI Universal Configuration Interface, et le package manager Opkg (anciennement Ipkg).

Des interfaces de configuration basées sur UCI permettent de configurer l'environnement OpenWRT graphiquement.

La plus connue est Luci, mais il en existe d'autres: X-wrt webif, Pyci ou encore Gargoyle.



UCI est un outil de centralisation de la configuration. l'ensemble de la configuration du système est centralisée dans le répertoire /etc/config, dans des fichiers plats.

L'outil UCI ou la librairie permet d'acceder à la configuration à partir d'une notation du style ‘MIB’ (config.section.key=value (à la SNMP))

Exemple de fichier de configuration UCI: network

```
# cat /etc/config/network
config interface loopback
        option ifname   lo
        option proto    static
        option ipaddr   127.0.0.1
        option netmask  255.0.0.0

config interface lan
        option ifname   eth0
        option type     bridge
        option proto    static
        option ipaddr   192.168.1.1
        option netmask  255.255.255.0
```

Utilisation de l'outil en ligne de commande **UCI**:

```
# uci show network.lan
network.lan=interface
network.lan.ifname=eth0
network.lan.type=bridge
network.lan.proto=static
network.lan.ipaddr=192.168.1.1
network.lan.netmask=255.255.255.0

# uci set network.lan.ipaddr=192.168.1.42

# uci get network.lan.ipaddr
192.168.1.42
```

Les mêmes fonctionnalités sont offertes par la librairie C libuci et libuci-lua.

Opkg est un package manager léger, minimaliste, qui ressemble un peu à dpkg/apt (Debian). Il est la continuité de Ipkg, dont il résoud de nombreux problèmes.

Opkg est destiné à l'utilisation sur plateformes embarquées et est actuellement utilisé par deux projets majeurs: OpenEmbedded et OpenWRT.

Fichier de configuration Opkg: */etc/opkg.conf*

```
src/gz packages http://downloads.openwrt.org/backfire/10.03/kirkwood/packages
dest root /
dest ram /tmp
lists_dir ext /var/opkg-lists
option overlay_root /overlay
```

```
# opkg help
opkg: unknown sub-command help
usage: opkg [options...] sub-command [arguments...]
where sub-command is one of:

Package Manipulation:
        update                  Update list of available packages
        upgrade <pkgs>          Upgrade package(s)
        install <pkgs>          Install package(s)
        configure <pkgs>        Configure unpacked package(s)
        remove <pkgs|regexp>    Remove package(s)
        flag <flag> <pkgs>      Flag package(s)
         <flag>=hold|noprune|user|ok|installed|unpacked (one per invocation)

Informational Commands:
        list                    List available packages
        list-installed          List installed packages
        list-upgradable         List installed and upgradable packages
        files <pkg>             List files belonging to <pkg>
        search <file|regexp>    List package providing <file>
        info [pkg|regexp]       Display all info for <pkg>
        status [pkg|regexp]     Display all status for <pkg>
        download <pkg>          Download <pkg> to current directory
        ...
```

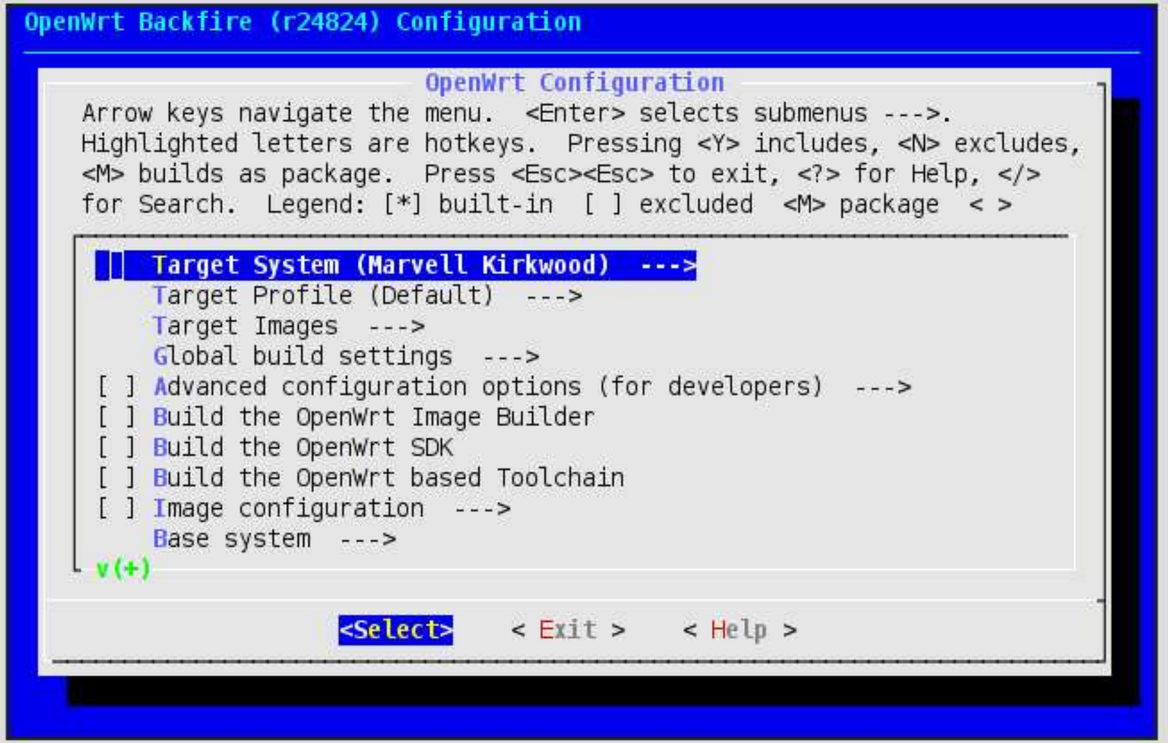
Luci, à l'origine une interface web de configuration d'un firmware OpenWRT, developpé en Lua en suivant l'architecture logicielle MVC, pour le projet Freifunk, a évolué pour devenir une partie officielle d'OpenWRT, qui fournie un ensemble de librairies (accès au système, configuration UCI, service JSON-RPC…) ainsi qu'une interface web de configuration.

```
$ ls libs/
core  fastindex  httpclient  ipkg  json  lmo lucid  lucid-http  lucid-rpc
nixio  px5g  rpcc  sgi-cgi  sgi-luci  sgi-uhttpd sgi-wsapi  sys  web
```

```
├── bin                    Firmware, kernel résultant de la compilation
├── BSDmakefile
├── build_dir              Répertoire où se passe les compilations
├── Config.in              Configuration du menu 'menuconfig'
├── dl                     Stockage des tarballs
├── docs
├── feeds                  Feeds de paquets (makefiles, etc)
├── feeds.conf.default     Liste de feeds officiels ou connus
├── include                Makefiles à inclure (modularisation)
├── LICENSE
├── Makefile
├── package                Paquets de base (information de packaging: makefiles, etc)
├── README
├── rules.mk               Makefile à inclure (modularisation)
├── scripts                Divers scripts utilisés par le build system
│                            (éventuellement à utiliser à la main)
├── staging_dir            Répertoire où git les binaires et autres fichiers
│                            déja traités (target, toolchain...)
├── target                 Makefiles spécifiques pour architectures (Linux: ARM, Mips, ...)
│                            ou application (SDK, imagebuilder...)
├── tmp
├── toolchain              Makefiles spécifiques à la toolchain
└── tools                  Makefiles des outils de build
```

à la racine des sources d'OpenWRT:

```
$ make menuconfig
```



Sélection de l'architecture et de la machine, des diverses options de compilation ou choix logiciel.

Puis:

```
$ make [-j2] [V=99]
```

Construit le firmware (rootfs, image jffs2 ou autre, kernel Linux) après, bien sur, la construction de la toolchain et des outils de build.

Les images du firmware et les paquets sont disponibles dans bin/:

```
$ ls bin/kirkwood/
md5sums  openwrt-kirkwood-jffs2-128k.img  openwrt-kirkwood-rootfs.tar.gz
openwrt-kirkwood-uImage  packages
```

Piocher dans les feeds

De base, seuls les packages de base sont disponibles. Pour pouvoir ajouter d'autres logiciels, il faut installer un “feed”, puis sélectionner les paquets dans l'interface de configuration ‘menuconfig’.

Le feed doit être présent dans feeds.conf (feed.conf.default contient les feeds officiels/connus).

```
$ cat feeds.conf.default
src-svn packages svn://svn.openwrt.org/openwrt/packages
src-svn xwrt http://x-wrt.googlecode.com/svn/branches/backfire_10.03/package
$ ./scripts/feeds update xwrt
$ ./scripts/feeds install -a -p xwrt
Installing all packages from feed xwrt.
Installing package 'm-route'
Installing package 'webif'
Installing package 'haserl'
Installing package 'webif-applications'
...
```

Ces nouveaux paquets sont maintenant disponibles dans l'interface ‘menuconfig’.

Une configuration optimale est disponible pour chaque plateforme supportée. Il peut cependant être souhaitable de modifier cette configuration soit même:

```
$ make kernel_menuconfig
```

Note: Pour l'ajout d'une fonctionnalité ou d'un driver, d'abord jeter un oeil à la section ‘kernel modules’ de l'interface de configuration du firmware (menuconfig) !

Cette présentation n'est bien sur pas exhaustive…

- Wiki OpenWRT
- Documentation Buildroot
- Manuel Uboot

Il est venu le temps, des rires et des chants, de la lecture de man et de howtos, d'expérimenter et de découvrir par nous même !

à nos machines !

- Guruplug server
- Sheevaplug
- Plug Computer Wiki

OpenWRT pour Sheevaplug/Guruplug

- Architecture ARM
- Plateforme kirkwood
- Installation: Flashage via Uboot, utilisation de la console série

Utilisez votre guruplug comme serveur de streaming: diffusez le flux vidéo capturé par la webcam.

- Construisez un firmware openwrt adapté au guruplug. Créez une image Ubifs (Attention, piège!).
- Installez le firmware sur la flash, en utilisant le système de fichier Ubifs. Configurez uboot pour booter sur openwrt.
- Installez un logiciel de streaming vidéo de votre choix (motion, uvc-streamer, mjpeg-streamer, gstreamer, vlc…)
- Streamez les progrets de vos voisins !

En plus pour les plus rapides:

- Ressortez un de vos projets personnels ou d'école en C, cross-compilez le pour le faire fonctionner sur le guruplug.
- créez votre feed et packagez ce projet. installez ensuite le paquet sur le guruplug

Déja fini?

- ajoutez la prise en charge de votre logiciel dans Luci. Si une configuration est nécessaire, elle doit être gérée par UCI.

De bon liens :)

- **Elinux**: Embeded Linux wiki, une **mine** d'informations
- **Documentation FreeElectron**, des présentations sur l'embarqué ou le développement kernel
- **Hackable-devices**: trouvez d'autres geeks/hackers, des idées de projets, les prochains évênements, du matériel libre et/ou bidouillable !
- **Reverse engineering de matériel embarqué**