



# Formation développement de systèmes Linux embarqué

Formation sur site, 5 jours

Dernière mise à jour : 20 May 2024

<b>Titre</b>	<b>Formation développement de systèmes Linux embarqué</b>
<b>Objectifs opérationnels</b>	<ul style="list-style-type: none"><li>• Être capable d'appréhender l'architecture générale d'un système Linux embarqué.</li><li>• Être capable de sélectionner, construire, mettre en oeuvre et utiliser une chaîne de compilation croisée.</li><li>• Être capable de comprendre la séquence d'un démarrage d'un système Linux embarqué et de mettre en oeuvre et d'utiliser le chargeur de démarrage U-Boot.</li><li>• Être capable de sélectionner une version du noyau Linux, de configurer, de compiler et d'installer le noyau Linux sur un système embarqué.</li><li>• Être capable de créer à partir de zéro un système de fichiers racine Linux, en comprenant les différents éléments qui le composent : répertoires, applications, bibliothèques, fichiers de configuration.</li><li>• Être capable de choisir et de mettre en oeuvre les principaux systèmes de fichiers Linux pour périphérique de stockage en mode bloc et flash, et de connaître leurs principales caractéristiques.</li><li>• Être capable d'interagir avec les périphériques matériels, de configurer le noyau avec les pilotes de périphériques nécessaires et d'étendre le <i>Device Tree</i></li><li>• Être capable de sélectionner, de cross-compiler et d'intégrer des composants logiciels open-source (bibliothèques, applications) dans un système Linux embarqué, et de traiter la mise en conformité avec les licences open-source.</li><li>• Être capable de mettre en oeuvre un système de build Linux embarqué, pour construire un système complet pour une plateforme embarquée.</li><li>• Être capable de développer et déboguer des applications sur un système Linux embarqué.</li></ul>
<b>Durée</b>	<b>Cinq</b> jours - 40 h (8 h par jour)
<b>Méthodes pédagogiques</b>	<ul style="list-style-type: none"><li>• Présentations animées par le formateur : 50% de la durée de formation</li><li>• Travaux pratiques réalisés par les participants : 50% de la durée de formation</li><li>• Version électroniques de supports de présentation, des instructions et des données de travaux pratiques. Les supports sont librement disponibles sur <a href="https://bootlin.com/doc/training/embedded-linux">https://bootlin.com/doc/training/embedded-linux</a>.</li></ul>



<b>Formateur</b>	Un des ingénieurs mentionnés sur : <a href="https://bootlin.com/training/trainers/">https://bootlin.com/training/trainers/</a>
<b>Langue</b>	Présentations : Français Supports : Anglais
<b>Public visé</b>	Ingénieurs développant des systèmes embarqués reposant sur Linux et des composants open-source.
<b>Pré-requis</b>	<ul style="list-style-type: none"><li>• <b>Connaissance et pratique des commandes UNIX ou GNU/Linux</b> : les participants doivent être à l'aise avec l'utilisation de la ligne de commande Linux. Les participants manquant d'expérience sur ce sujet doivent se former par eux-mêmes, par exemple en utilisant nos supports de formation disponible à l'adresse <a href="https://bootlin.com/blog/command-line/">bootlin.com/blog/command-line/</a>.</li><li>• <b>Niveau minimal requis en anglais : B1</b>, d'après le <i>Common European Framework of References for Languages</i>, pour nos sessions animées en anglais. Voir <a href="https://bootlin.com/pub/training/cefr-grid.pdf">bootlin.com/pub/training/cefr-grid.pdf</a> pour une auto-évaluation.</li></ul>
<b>Équipement nécessaire</b>	<b>Pour les sessions en présentiel dans les locaux de nos clients, notre client doit fournir :</b> <ul style="list-style-type: none"><li>• Projecteur vidéo</li><li>• Un ordinateur sur chaque bureau (pour une ou deux personnes), avec au moins 8 Go de RAM et Ubuntu Linux 22.04 installé dans une <b>partition dédiée d'au moins 30 Go</b>.</li><li>• Les distributions autres que Ubuntu Linux 22.04 ne sont pas supportées, et l'utilisation de Linux dans une machine virtuelle n'est également pas supportée.</li><li>• <b>Connexion à Internet rapide et sans filtrage</b> : au moins 50 Mbit/s de bande passante en téléchargement, et pas de filtrage des sites Web et protocoles.</li><li>• <b>Les ordinateurs contenant des données importantes doivent être sauvegardés</b> avant d'être utilisés dans nos sessions.</li></ul>
<b>Modalités d'évaluation</b>	Seuls les participants qui auront assisté à l'intégralité des journées de formation, et qui auront obtenu plus de 50% de réponses correctes à l'évaluation finale recevront une attestation individuelle de formation de la part de Bootlin.



## Handicap

Les participants en situation de handicap qui ont des besoins spécifiques sont invités à nous contacter à l'adresse [training@bootlin.com](mailto:training@bootlin.com) afin de discuter des adaptations nécessaires à la formation.

## Plateforme matérielle pour les travaux pratiques, option #1

Une de ces cartes de STMicroelectronics :  
**STM32MP157A-DK1**, **STM32MP157D-DK1**,  
**STM32MP157C-DK2** ou **STM32MP157F-DK2**

- Processeur STM32MP157, double Cortex-A7, de STMicroelectronics
- Alimentée par USB
- 512 Mo DDR3L RAM
- Port Gigabit Ethernet port
- 4 ports hôte USB 2.0
- 1 port USB-C OTG
- 1 connecteur Micro SD
- Debugger ST-LINK/V2-1 sur la carte
- Connecteurs compatibles Arduino Uno v3
- Codec audio
- Divers : boutons, LEDs
- Écran LCD tactile (uniquement sur cartes DK2)

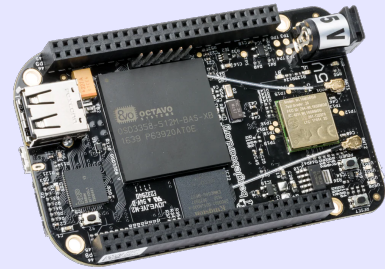




## Plateforme matérielle pour les travaux pratiques, option #2

### Carte **BeagleBone Black** ou **BeagleBone Black Wireless**

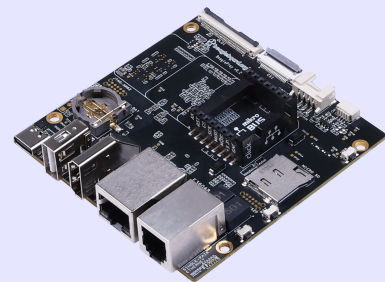
- Un processeur ARM AM335x de Texas Instruments (à base de Cortex-A8), avec accélération 3D, etc.
- 512 Mo de RAM
- 2 ou 4 Go de stockage eMMC
- USB hôte et device
- Sortie HDMI
- Connecteurs à 2 x 46 broches, pour accéder aux UARTs, aux bus SPI, aux bus I2C, et à d'autres entrées/sorties du processeur.
- Ethernet ou WiFi



## Plateforme matérielle pour les travaux pratiques, option #3

### Carte **BeaglePlay**

- SoC Texas Instruments AM625x (CPU 4xARM Cortex-A53)
- SoC avec accélération 3D, MCU intégré et de nombreux autres périphériques.
- 2 GB de RAM
- 16 Go de stockage eMMC
- USB hôte et device, microSD, HDMI
- WiFi 2.4 and 5 GHz, Bluetooth et aussi Ethernet
- 1 Header MicroBus (SPI, I2C, UART, ...), connecteurs OLDI et CSI.





## 1<sup>er</sup> jour - Matin

### Cours – Introduction à Linux embarqué

- Introduction au Logiciel Libre
- Atouts du Logiciel Libre pour les systèmes embarqués
- Exemples de systèmes embarqués fonctionnant sous Linux
- Besoins en CPU, RAM et stockage
- Choix d'une plateforme matérielle
- Architecture du système : composants principaux
- Différentes tâches pour développer un système embarqué

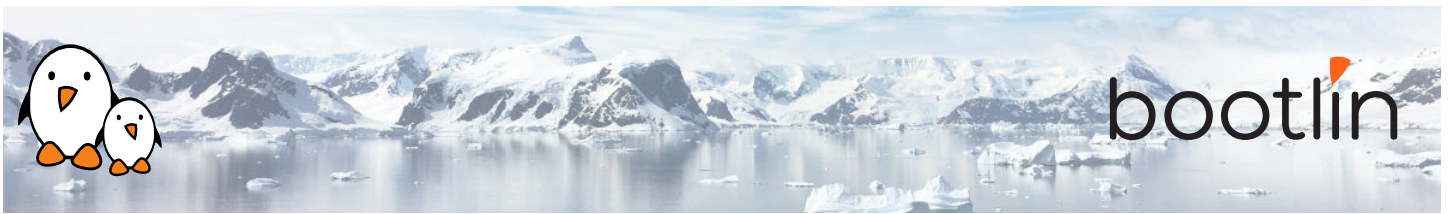
### Cours - Chaîne de compilation croisée et bibliothèque standard C

- Les composants d'une chaîne de compilation croisée.
- Choisir une bibliothèque standard C.
- Le contenu de la bibliothèque standard C.
- Les chaînes de compilation croisée prêtes à l'emploi.
- La construction automatisée d'une chaîne de compilation croisée.

### TP – Chaîne de compilation croisée

- Configuration de Crosstool-NG
- Exécution pour construire une chaîne de compilation croisée personnalisée reposant sur la uClibc.
- Exploration du contenu d'une chaîne de compilation croisée





## 1<sup>er</sup> Jour - Après-midi

### Cours – Processus de démarrage, firmware, chargeurs de démarrage

- Processus de démarrage des systèmes embarqués, focus sur les architectures *x86* et *ARM*
- Processus de démarrage et chargeurs de démarrage sur plateformes *x86* (legacy et UEFI)
- Processus de démarrage sur plateformes *ARM* : code en ROM, chargeurs de démarrage, *ARM Trusted Firmware*
- Focus sur U-Boot : configuration, installation et utilisation
- Commandes U-Boot, environnement U-Boot, scripts U-Boot, mécanisme *distro boot command* de U-Boot

### TP - U-Boot

- Mise en place de la communication série avec la carte.
- Configuration, compilation et installation d'U-Boot sur la plateforme embarquée.
- Seulement sur STM32MP1 : Configuration, compilation et installation de Trusted Firmware-A sur la plateforme embarquée.
- Familiarisation avec l'environnement et les commandes d'U-Boot.
- Mise en place de la communication TFTP avec la carte. Utilisation des commandes TFTP d'U-Boot.

*Mise en oeuvre sur la plateforme embarquée.*

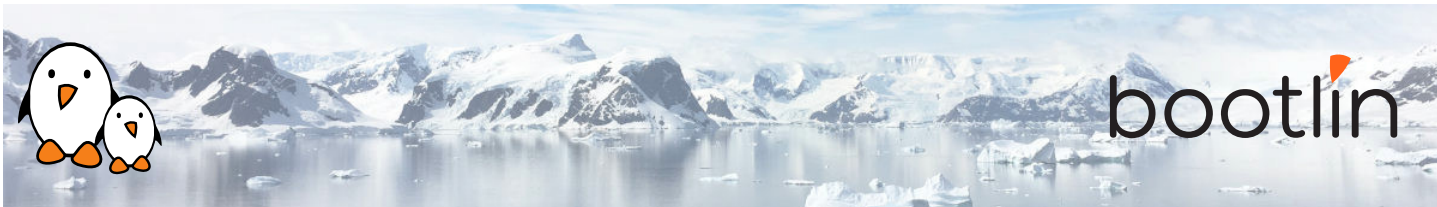
## 2<sup>ème</sup> jour - Matin

### Cours – Noyau Linux

- Rôle et architecture générale du noyau Linux.
- Séparation entre noyau et espace utilisateur, interfaces entre l'espace utilisateur et le noyau
- Comprendre les différentes versions du noyau Linux : choix entre versions proposées par les fabricants et la version *upstream*, versions *Long Term Support*
- Récupérer les sources du noyau Linux

### Lab - Récupération des sources du noyau Linux

- Clonage du dépôt git officiel de Linux
- Accès aux versions stables



### Cours – Configuration, compilation et démarrage du Noyau Linux

- Configuration du noyau Linux : configuration pré-définies, interfaces de configuration
- Concept de *Device Tree*
- Cross-compilation du noyau Linux
- Rôle des fichiers résultants de la compilation du noyau Linux
- Installation et démarrage du noyau Linux
- La ligne de commande du noyau Linux

### TP - Compilation croisée du noyau et démarrage sur la carte

- Configuration du noyau Linux et compilation croisée pour la plateforme embarquée.
- Téléchargement du noyau en utilisant le client TFTP d'U-Boot.
- Démarrage du noyau depuis la RAM.
- Automatisation du démarrage du noyau avec des scripts U-Boot.

*Mise en oeuvre sur la plateforme embarquée.*

## 2<sup>ème</sup> jour - Après-midi

### Cours – Système de fichier racine

- Les systèmes de fichiers dans Linux.
- Rôle et organisation du système de fichiers racine.
- Localisation de ce système de fichiers : sur espace de stockage, en mémoire, sur le réseau.
- Les fichiers device, les systèmes de fichiers virtuels.
- Contenu type d'un système de fichiers racine.

### Cours - BusyBox

- Présentation de BusyBox. Intérêt pour les systèmes embarqués.
- Configuration, compilation et installation.



## TP – Construction d’un minuscule système Linux embarqué avec BusyBox

- Construction à partir de zéro d’un système de fichiers racine contenant un système Linux embarqué
- Mise en place d’un noyau permettant de démarrer le système depuis un répertoire mis à disposition par la station de développement au travers de NFS.
- Passage de paramètres au noyau pour le démarrage avec NFS.
- Création complète du système de fichiers à partir de zéro : installation de BusyBox, création des fichiers spéciaux pour les périphériques.
- Initialisation du système en utilisant le programme init de BusyBox.
- Utilisation du serveur HTTP de BusyBox.
- Contrôle de la cible à partir d’un navigateur Web sur la station de développement.
- Mise en place des bibliothèques partagées sur la cible et développement d’une application d’exemple.

*Mise en oeuvre sur la plateforme embarquée.*

## 3<sup>ème</sup> jour - Matin

### Cours - Accès aux périphériques matériels

- Comment accéder au matériel sur les principaux bus : USB, SPI, I2C, PCI
- Utilisation de pilotes de périphériques dans le noyau ou accès depuis l’espace utilisateur
- La syntaxe du *Device Tree*, et comment l’utilisation pour décrire des périphériques additionnels et le multiplexage des signaux.
- Trouver des pilotes de périphériques dans le noyau Linux pour des périphériques matériels.
- Utilisation de modules noyau
- Accès au matériel par `/dev` ou `/sys`
- Interfaces en espace utilisateur pour les périphériques les plus courants : stockage, réseau, GPIO, LEDs, audio, affichage, video

### TP - Accès aux périphériques matériels

- Exploration du contenu de `/dev` et `/sys`, et des périphériques disponibles sur la plateforme embarquée.
- Utilisation de GPIOs et de LEDs.
- Modification du Device Tree pour contrôler le multiplexage de broches et déclarer un joystick connecté sur I2C.
- Ajout du support pour une carte son USB en utilisant des modules noyau.
- Prise en charge du joystick I2C par la compilation et l’installation d’un module noyau externe.

*Mise en oeuvre sur la plateforme embarquée.*





## 3<sup>ème</sup> jour - Après-midi

### Cours - Système de fichiers bloc

- Accéder et partitionner des périphériques bloc.
- Systèmes de fichiers pour périphériques bloc.
- Utilité des systèmes de fichiers journalisés.
- Systèmes de fichiers en lecture seule.
- Systèmes de fichiers en RAM.
- Création de chacun de ces systèmes de fichiers.
- Suggestions pour les systèmes embarqués.

### TP - Système de fichiers bloc

- Créer des partitions sur le stockage bloc.
- Démarrage d'un système avec un assemblage de plusieurs systèmes de fichiers : SquashFS pour le système de fichier racine, ext4 pour les données du système et tmpfs pour les fichiers temporaires.

*Mise en oeuvre sur la plateforme embarquée.*

### Cours - Système de fichiers pour flash

- Le sous-système Memory Technology Devices du noyau Linux.
- Les systèmes de fichiers pour le stockage MTD : JFFS2, YAFFS2, UBIFS.
- Options de configuration du noyau.
- Partitions MTD.
- Étude en détail de UBI et UBIFS : préparation, flashage et mise en oeuvre d'images UBI.

*Note : la plateforme embarquée utilisée pour les labs ne comportant pas de mémoire flash NAND en accès direct, cette partie du cours ne sera pas illustrée avec un TP correspondant.*



## 4<sup>ème</sup> jour - Matin

### Cours – Cross-compilation de bibliothèques et d'applications espace utilisateur

- Configuration, compilation croisée et installation d'applications et de bibliothèques
- Concept de *build system*, et aperçu de quelques *build systems* courants utilisés dans les projets open-source : *Makefile*, *autotools*, *CMake*, *meson*
- Aperçu des problématiques courantes rencontrées lors de la compilation croisée

### TP – Compilation croisée de bibliothèques et d'applications

- Compilation croisée manuelle de plusieurs bibliothèques et applications open-source pour une plateforme embarquée.
- Apprentissage des principales techniques et des problèmes principaux.
- Cela inclut la compilation du composant *alsa-utils*, et l'utilisation de son programme *speaker-test* pour vérifier que l'audio fonctionne sur la cible.

*Mise en oeuvre sur la plateforme embarquée.*

## 4<sup>ème</sup> jour - Après-midi

### Cours - Outils de construction de systèmes

- Les différentes approches pour construire un système Linux embarqué : *build systems* et distributions binaires
- Principe d'un *build system*, aperçu de Yocto Project/OpenEmbedded et de Buildroot
- Principe d'une *distribution binaire* et outils associés, focus sur Debian et Ubuntu
- Piles logicielles standards : Tizen, AGL, Android

### TP - Construction d'un système avec Buildroot

- Utilisation de Buildroot pour construire de façon automatisée le même système de base que dans le TP précédent, en y rajoutant un serveur pour jouer des bandes sonores (*MPD*) ainsi qu'un client pour piloter ce serveur (*mpc*).
- Contrôle de la lecture audio, directement depuis la cible, puis depuis un client MPD sur le PC de développement.
- Analyse des dépendences entre les différents composants du système.

*Mise en oeuvre sur la plateforme embarquée.*



## Cours - Licences open-source et mise en conformité

- Présentation des principales licences open-source : GPL, LGPL, MIT, BSD, Apache, etc.
- Concept de *copyleft* dans les licences open-source
- Différence entre (L)GPL version 2 et 3
- Mise en conformité avec les licences open-source : bonnes pratiques

## 5<sup>ème</sup> jour - Matin

### Cours - Aperçu des stack logicielles open-source majeures pour Linux embarqué

- `systemd` comme système *init*
- Gestion du matériel avec `udev`
- Communication inter-processus avec `D-Bus`
- La stack logicielle pour le graphique : DRM/KMS, X.org, Wayland, Qt, Gtk, OpenGL
- La stack logicielle pour le multimédia : Video4Linux, GStreamer, Pulseaudio, Pipewire

### TP - Intégration de stack logicielles additionnelles

- Intégration de `systemd` comme système d'*init*
- Utilisation de `udev` pour le chargement automatique de modules

*Mise en oeuvre sur la plateforme embarquée.*



## 5<sup>ème</sup> jour - Après-midi

### Cours - Développement et débogage d'application

- Langages de programmations et bibliothèques disponibles.
- *Build system* pour votre application, un aperçu de *CMake* et *Meson*
- Le débogueur *gdb* : débogage d'applications à distance avec *gdb* et *gdbserver*, analyse post-mortem d'une application.
- Analyse de performance, outils de *tracing* et *profiling*, analyseurs mémoire : *strace*, *ltrace*, *perf*, *valgrind*

### TP – Développement et débogage d'application

- Création d'une application qui utilise un joystick connecté sur I2C pour contrôler un player audio.
- Mise en place d'un IDE pour le développement et le débogage d'une application.
- Utilisation de *strace*, *ltrace*, *gdbserver* et *perf* pour déboguer/investiguer des applications problématiques sur la plateforme embarquée.

*Mise en oeuvre sur la plateforme embarquée.*

### Lecture - Ressources utiles

- Livres sur Linux embarqué et la programmation système
- Ressources sur Internet
- Conférences internationales