D1 - Linux embarqué avec Buildroot et Yocto

Construire et installer une plateforme Linux pour système embarqué

Objectifs

- Comprendre l'architecture d'un système Linux
- Créer et utiliser une chaîne de compilation croisée
- Apprendre à installer Linux sur votre cible matérielle et écrire un BSP
- Installer Xenomai pour le temps réel
- Explorer l'architecture système de Linux
  - Boot de Linux
  - Initialiser le système
- Installer des paquetages logiciels existant sur la cible
- Apprendre à flasher Linux

Les exercices se font sur des cartes cibles :

- *STM32MP15-DISCO basée sur un ARM Cortex/A7 dual cœur.*
- *NXP i.MX6 Sabrelite basée sur un ARM Cortex/A9 quadri cœur.*
- *NXP i.MX8MQ-EVK basée sur un ARM Cortex/A53 quadri cœur.*

Nous utilisons le dernier noyau supporté par le fournisseur du chip (4.x)

Matériel

- Un PC Linux par binôme
- Une carte embarquée par binôme
- Support de cours fournis

Environnement du cours

- Cours théorique
  - Support de cours imprimé et au format PDF (en anglais).
  - Le formateur répond aux questions des stagiaires en direct pendant la formation et fournit une assistance technique et pédagogique.
- Activités pratiques
  - Les activités pratiques représentent de 40% à 50% de la durée du cours.
  - Elles permettent de valider ou compléter les connaissances acquises pendant le cours théorique.
  - Exemples de code, exercices et solutions
  - Un PC (Linux ou Windows) par binôme de stagiaires (si plus de 6 stagiaires) pour les activités pratiques avec, si approprié, une carte cible embarquée.
  - Le formateur accède aux PC des stagiaires pour l'assistance technique et pédagogique.
- Une machine virtuelle préconfigurée téléchargeable pour refaire les activités pratiques après le cours
- Au début de chaque demi-journée une période est réservée à une interaction avec les stagiaires pour s'assurer que le cours répond à leurs attentes et l'adapter si nécessaire

Pré-requis

- Bonne maîtrise du langage C
- Connaissance de la programmation Linux en mode utilisateur (voir notre cours D0 - Programmation en mode utilisateur Linux)
- De préférence, connaissance du noyau Linux et de la programmation de driver (voir notre cours D3 - Drivers Linux)

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus

Audience visée

- Tout ingénieur ou technicien en systèmes embarqués possédant les prérequis ci-dessus

Modalités d'évaluation

- Les prérequis indiqués ci-dessus sont évalués avant la formation par l'encadrement technique du stagiaire dans son entreprise, ou par le stagiaire lui-même dans le cas exceptionnel d'un stagiaire individuel.
- Les progrès des stagiaires sont évalués de deux façons différentes, suivant le cours:
  - Pour les cours se prêtant à des exercices pratiques, les résultats des exercices sont vérifiés par le formateur, qui aide si nécessaire les stagiaires à les réaliser en apportant des précisions supplémentaires.
  - Des quizz sont proposés en fin des sections ne comportant pas d'exercices pratiques pour vérifier que les stagiaires ont assimilé les points présentés
- En fin de formation, chaque stagiaire reçoit une attestation et un certificat attestant qu'il a suivi le cours avec succès.
  - En cas de problème dû à un manque de prérequis de la part du stagiaire, constaté lors de la formation, une formation différente ou complémentaire lui est proposée, en général pour conforter ses prérequis, en accord avec son responsable en entreprise le cas échéant.

Plan

1er jour

Architecture de Linux

- Linux
  - Histoire
  - Gestion de version
- Les diverses licences utilisées par Linux (GPL, LGPL, etc)
- Distributions Linux
- Architecture et modularité de Linux

Les chaînes de compilation croisée

- Chaînes de compilation croisée pré-compilées
- Outils de génération de chaînes de compilation croisée
  - Crosstool-ng
  - Buildroot
- Compilation manuelle de chaîne de compilation croisée

Exercise : *Construction d'une chaîne de compilation croisée avec Crosstool-ng*

Les outils Linux pour l'embarqué

- Les bootloaders (Uboot, Redboot, barebox)
- Les librairies adaptée à l'embarqué (eglibc, uClibc)
- Les IHM adaptées à l'embarqué
- Busybox

Le boot loader U-Boot

- Introduction à U-Boot
- Booter la carte à travers U-Boot
  - Booter depuis la NOR

    - Booter depuis la NAND
    - Booter depuis la eMMC
- Variables d'environnement d'U-Boot
    - Variables définies par l'utilisateur
    - Variables prédéfinies
    - Substitution de variable
- Le shell U-Boot minimal
    - Ecrire des scripts dans des variables
    - Exécuter des scripts
    - Utiliser des variables dans des scripts : le patron set-script
- Principales commandes d'U-Boot
- Booter un OS
    - Accéder aux flashs
    - Accéder aux systèmes de fichier (NFS, FAT, EXTx, JFFS2&)
- Le shell U-Boot complet
    - Structure du script
    - Instructions de contrôle (if, for&)

Exercise : *Ecrire un script qui configure le réseau et passe cette information au noyau Linux*
Exercise : *Booter la carte en NFS, en utilisant des images pré-existantes*
Exercise : *Ecrire des scripts pour choisir entre booter depuis la flash ou le réseau*

2ème jour

Créer le noyau Linux embarqué

- Télécharger un code source stable
    - Obtenir une tarball
    - Utiliser GIT
- Configurer le noyau
- Compiler le noyau et ses modules
    - Modules internes aux sources de Linux (in-tree)
    - Modules externes aux sources (out-of-tree)
- Installer le noyau et les modules

Exercise : *Configurer et compiler un noyau pour la carte cible*

Le BSP Linux

- Architecture du BSP Linux
    - Structure générale
    - Le BSP ARM
    - Le système de compilation de Linux
- Définir et initialiser la carte
    - Programmatiquement (platform, i2c, spi, &)
    - En utilisant le Flattened Device Tree

Exercise : *Créer un BSP minimal pour la carte cible*

Créer un système de fichier racine

- Paquetages
    - Divers systèmes de compilation de paquetages (autotools, CMake, &)
    - Compiler un paquetage en croisée
- Les applications tout-en-un
    - Busybox, les utilitaires basiques
    - Dropbear: communications cryptées(ssh)
- Construire manuellement son système de fichier racine
    - Fichiers de périphérique, programmes et bibliothèques
    - Fichiers de configuration (réseau, udev, &)
    - Installer des modules
    - Chercher et installer les bibliothèques dont on a besoin

- Tester la cohérence et la complétude du système de fichier

Exercise : *Configurer et compiler Busybox et Dropbear*
Exercise : *Créer un système de fichier racine minimal en utilisant Busybox et Dropbear*

Le boot de Linux

- Paramètres du noyau Linux
- La séquence de démarrage de Linux
- Divers systèmes d'initialisation (busybox init, system V init, systemd)
- Démarrer automatiquement un système embarqué

Exercise : *Booter Linux en démarrant automatiquement une application utilisateur*

3ème jour

Systèmes de fichier embarqués

- Interfaces de stockage
  - Péripéhrique bloc
  - MTD
- Mémoires flash et MTDs Linux
  - flash NOR
  - flash NAND
  - flash ONENAND
- Les divers formats de système de fichier pour flash
  - JFFS2, YAFFS2, UBIFS
- Systèmes de fichier en lecture seule
  - CRAMFS, SQUASHFS
- Systèmes de fichier standards de Linux
  - Ext2/3/4, FAT, NFS
- Ramdisks et initrd
  - Créer un initramfs
  - Booter à travers un initramfs
- Choisir les bons formats de système de fichier
- Flasher le système de fichier

Exercise : *Construire un système de fichier racine de type initrd*

Buildroot

- Fonctionnement
  - Configuration de la chaîne de compilation
  - Sélection de paquetages
  - Configuration système (port série, remplissage de /dev, &)
  - Configuration du noyau et du boot-loader
  - Construire une image de système de fichier
- Adaptation
  - Utiliser une chaîne de compilation pré-compilée
  - Ajouter un patch à un paquetage existant
  - Ajouter un nouveau paquetage
  - Utiliser un squelette de rootfs customisé

Exercise : *Construire un système de fichier racine avec Buildroot*

4ème jour

Introduction à Yocto

- Présentation de Yocto
  - Histoire
  - Yocto, Open Embedded et Poky

    - Objetif du projet Yocto
    - Les principaux projets
- Architecture Yocto
    - Aperçu
    - Recettes et classes
    - Les tâches

Exercise : *Construire un système de fichier racine avec Yocto*

Le système de compilation Yocto

- Objectifs du système de compilation
    - Construire des images déployables
    - Couches et priorités de couches
    - Mise en page de l'annuaire
    - layout des dossiers
    - Fichiers de configuration (local, machine et distribution)
    - L'outil bitbake
- Utiliser Yocto
    - Build d'un package
    - Build d'une image (rootfs, u-boot et noyau)

Exercise : *Utiliser les commandes bitbake pour compiler des packages et des images*

Structure des recettes du package Yocto

- Architecture de recettes
    - Les tâches
    - Dépendances de tâches
    - Dépendances de recettes
- Le langage bitbake
    - Variables et fonctions standard
    - Les classes et les recettes
    - La classe « base »
    - Les commandes principales de bitbake
- Ajouter un nouveau layer
    - Structure de la couche (Layer)
    - Différents types de couches (Layer)

Exercise : *Ajouter un nouveau layer*
Exercise : *Ajouter une nouvelle recette*

Annexes

Linux temps-réel

- solutions temps-réel pour Linux
    - xenomai
    - patch rt preempt
- architecture de Xenomai
    - co-kernel
    - skins
    - drivers RTDM
- installation
    - installer Xenomai
    - cross-compiler une application Xenomai

Exercise : *installer Xenomai 3.x*
Exercise : *mesurer les temps de latence*
Exercise : *cross-compiler une application Xenomai*

Le BSP U-Boot

- Portage de U-Boot
  - Structure de code source U-Boot
  - Ajouter une nouvelle carte aux sources U-Boot
  - Drivers U-Boot (RS232, réseau, flash, SD / MMC)
  - Démarrage U-Boot et initialisation de la carte

Exercise : *Création d'un BSP dans U-Boot et configuration du code d'initialisation*

Renseignements pratiques

Durée : 4 jours
Prix : 2490 € HT