

---

# **UBS "Documentation du Mini projet : Home Automation Box"**

*Version 0.1*

**JAVIER LÓPEZ José Antonio**

02 January 2019



<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Développement</b>	<b>3</b>
2.1	Configuration du système pour la machine cible à utiliser . . . . .	3
2.2	Creation et configuration d'un layer et une recette image . . . . .	3
2.3	Configuration du local.conf : . . . . .	4
2.4	Compilation et test de l'image . . . . .	5
2.4.1	Compilation de l'image . . . . .	5
2.4.2	Test de l'image sur QEMU . . . . .	6
2.4.3	Test de l'image sur la machine cible . . . . .	6
2.5	Création et compilation d'une application . . . . .	7
2.5.1	Création d'une application . . . . .	7
2.5.2	Compilation d'une application . . . . .	9
2.6	Création et initialisation d'un service . . . . .	10
2.7	Démarrage d'un distribution Linux via Emmc . . . . .	11
<b>3</b>	<b>Resultats</b>	<b>13</b>
<b>4</b>	<b>Conclusions</b>	<b>15</b>
<b>5</b>	<b>Références</b>	<b>17</b>



---

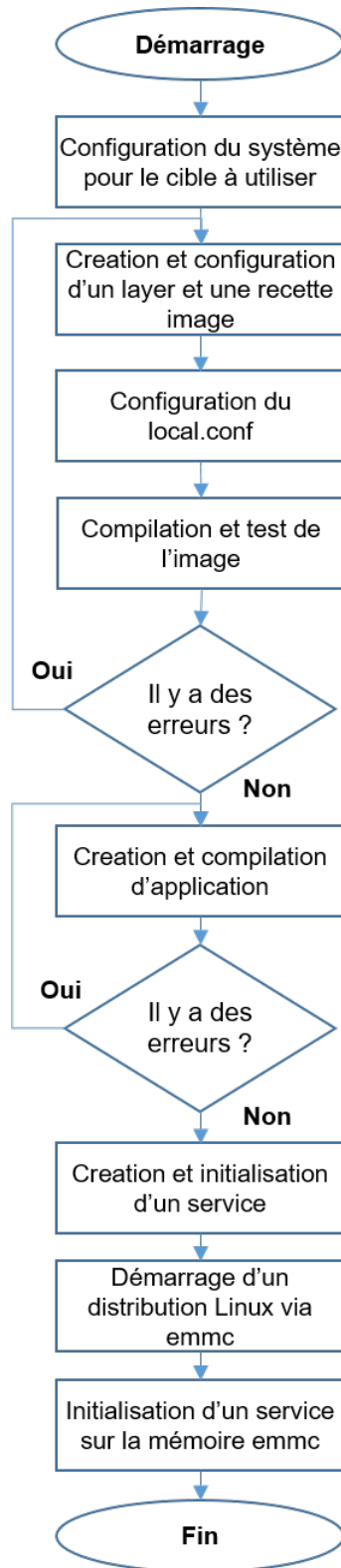
# Introduction

---

Le but de cette page sera d'expliquer tous les démarches pour créer une distribution linux pour la machine cible "humingoboard HB2 Gate 1.4". Ainsi que la détection de l'humidité et la temperature de l'environnement en utilisant le capteur "Adafruit Si70212".

L'outil principal utilisé dans l'élaboration du projet a été l'outil "Yocto Project". Puisqu'il est "open source" et permet aux développeurs de créer des systèmes personnalisés basés sur Linux quelle que soit l'architecture matérielle. La distribution utilisée du Yocto project a été "Poky" car elle contient le système de construction OpenEmbedded (BitBake et OpenEmbedded Core) ainsi qu'un ensemble de "metadata", ce qui nous permet de faire notre propre distribution. La fonction principale qui a été utilisé de la distribution "Poky" a été BitBake puisqu'il est le cœur du système de compilation OpenEmbedded et il est responsable de l'analyse des "metadata", pour générer une liste de tâches qui seront exécutés.

Les étapes qui ont été suivies pour le développement du projet sont alors :



---

## Développement

---

### Configuration du système pour la machine cible à utiliser

Un layer est un dossier qui contient des ensembles d'instructions, ils indiquent au système de construction OpenEmbedded ce qu'il faut faire pour utiliser la machine cible "Humingoboard HB2 Gate 1.4".

La machine cible utilise le système Freescale i.MX6 System-on-a-Chip. Le layer BSP de ce SoC est le "méta-freescale", il contient par exemple des bibliothèques GPU, VPU, l'u-boot et Linux Kernel. Le layer "meta-solidrun", ce lui qui donne le support nécessaire pour les plateformes SolidRun's iMX6.

Pour ajouter les layers, il faudra les télécharger, les installer, les placer sur la même branche, initialiser l'environnement OpenEmbedded et finalement les attacher à l'aide de la commande Bitbake. Les lignes de commandes suivantes ont été utilisées :

```
$ git clone git ://git.yoctoproject.org/meta-freescale
$ cd meta-freescale
$ git checkout -b rocko origin/rocko
$ cd ..
$ git clone git://github.com/SolidRun/meta-solidrun-arm-imx6.git
$ cd meta-solidrun-arm-imx6
$ git checkout -b rocko origin/rocko
$ cd ..
$ source oe-init-build-env
$ bitbake-layers add-layer ../meta-freescale
$ bitbake-layers add-layer ../meta-solidrun-arm-imx6
```

### Creation et configuration d'un layer et une recette image

L'idée du projet est de générer notre propre distribution avec notre propre configuration, il faut donc créer une recette de type image. Nous partons de la basse que nous avons déjà initialisé l'environnement (source oe-init-build-env) et nous somme sur le dossier /poky. Les lignes de commandes suivantes ont été exécutées :

```
$ yocto-layer create ubs-homeautomation
Please enter the layer priority you'd like to use for the layer: [default: 6]
Would you like to have an example recipe created? (y/n) [default: n] n
Would you like to have an example bbappend file created? (y/n) [default: n] n
$cd ../meta-ubs-homeautomation
$mkdir -p recipes-core/images
$cd recipes-core/images
$touch image-ubs-homeautomation.bb
```

Les commandes précédentes nous permettent de créer notre propre layer sans exemples. Nous avons aussi créé le dossier “recipes-core/image” pour indiquer quelle recette sera compilée. Puis nous avons créé un fichier “image-ubs-homeautomation.bb” ce qui corresponde à notre fichier recette.

Finalement une série de modifications a été pris en compte afin de personnaliser notre distribution. Pour modifier tous les fichiers du projet, l’éditeur nano a été utilisé, puisqu’il est considéré comme un des éditeurs de texte plus simple à utiliser. La commande qui nous permet de modifier notre recette est alors :

```
$nano core-image-minimal.bb
```

Une fois en mode édition, les critères suivants ont été ajoutés :

- Ajout d’une image qui nous permet le démarrage de notre machine cible.

```
include recipes-core/images/core-image-minimal.bb
```

- Ajout d’un service qui nous permet la connexion à distance afin de mettre à jour le système ou réaliser différentes manipulations.

```
IMAGE_FEATURES += "ssh-server-dropbear"
```

- Ajout d’un service qui nous permet d’utiliser les GPIO de la carte, afin de se communiquer via I2C. Nous pourrions de cette façon établir une connexion avec le capteur “Adafruit Si7021”.

```
IMAGE_INSTALL += " i2c-tools"
```

- Ajout d’un compte utilisateur (“user : ubs” et “password :ssi”), en utilisant la classe extra user.

```
inherit extrausers
EXTRA_USERS_PARAMS = "useradd -P 'ssi' ubs;"
```

Un fois la configuration a été finalisé sur l’éditeur nano, les commandes “ctrl+o” et “ctrl+x” nous permettent d’enregistrer le fichier et de sortir de l’interface d’édition.

## Configuration du local.conf :

Pour optimiser et exécuter la compilation de notre distribution certains modifications sur le fichier “local.conf” ont été nécessaires. Les configurations suivantes ont été réalisées :

- La sélection d’une machine est une configuration nécessaire pour construire les paquets de l’architecture sélectionnée. Pour faire une compilation de notre machine cible, l’option “solidrun-imx6” doit être utilisée. Cependant l’option “qemuarm” a aussi été utilisée, afin de tester l’image sous un environnement virtuelle.

```
MACHINE ?= "qemuarm"
#MACHINE ??= "solidrun-imx6"
```

- Afin d’optimiser la compilation, le plus important est la suppression des sources après la compilation. L’option suivante a donc été ajoutée :

```
INHERIT += "rm_work" a
```

- Nous pouvons exécuter la compilation de forme parallèle, la diminution de temps est par suite considérable. Cependant le nombre de tâches parallèles à exécuter à une relation avec le nombre de processeurs disponible sur l’ordinateur. L’option suivante a donc été ajoutée :

```
PARALLEL_MAKE ?= "-j 4"
```

- Comme précédemment, nous pouvons définir le nombre de tâches que BitBake peut exécuter en parallèle. Ce processus permet à BitBake de diviser la construction des paquets en plusieurs tâches. L’option suivante a donc été ajoutée :



```
BB_NUMBER_THREADS ?= "4"
```

- Nous avons choisi l'utilisation du Package Management System, puisqu'il est un système de gestion de paquets léger conçu pour les systèmes embarqués. Le format IPK a l'avantage d'être très compact, bien plus que les deux autres formats RPM et DEB. Ce format permet aussi l'installation de nouveaux paquets, la mise à jour ou la suppression de paquets existants, l'interrogation des dépôts de paquets pour les paquets disponibles et la liste des paquets déjà installés. La configuration suivante a donc été utilisée :

```
PACKAGE_CLASSES ?= "package_ipk"
```

- Pour exécuter une application au démarrage, l'utilisation de Systemd est nécessaire. Systemd offre une meilleure gestion des services, il permet le chargement en parallèle des services au démarrage et il peut aussi diminuer le temps de démarrage du système. La configuration suivante a alors été utilisée :

```
DISTRO_FEATURES_append = " systemd"  
VIRTUAL-RUNTIME_init_manager = "systemd"
```

Afin d'éviter des problèmes au démarrage, nous avons empêché l'activation automatique de la fonction de distribution SysVinit, aussi nous avons supprimé tous les scripts SysVinit redondants, en utilisant la configuration suivante :

```
DISTRO_FEATURES_BACKFILL_CONSIDERED = "sysvinit"  
VIRTUAL-RUNTIME_initscripts = "
```

- Finalement pour accélérer les compilations futures, nous avons utilisé la fonction L\_DIR, puisque la fonction définit où les paquets téléchargés sont stockés. Si nous faisons une autre compilation, il n'essaiera pas de télécharger un paquet s'il est déjà présent. La configuration suivante a donc été utilisée :

```
DL_DIR ?= "${TOPDIR}/downloads"
```

Les réglages ont été effectués à l'aide de l'éditeur Nano. Cependant, n'importe quel éditeur peut être utilisé pour ajouter les paramètres.

## Compilation et test de l'image

### Compilation de l'image

Nous partons de la base que nous avons déjà initialisé l'environnement (source oe-init-build-env) et nous sommes sur le dossier poky/build/. Pour exécuter la compilation de notre image, la commande suivante a été utilisée :

```
$bitbake core-image-ubs-homeautomation
```

---

**Note :** Le processus de compilation peut prendre entre 5 et 6 heures, mais cela peut dépendre de la configuration effectuée dans le fichier "local.conf" ainsi que des caractéristiques de la machine hôte.

---

## Test de l'image sur QEMU

Lorsque le processus de compilation est terminé, l'image du système est prête et peut être utilisée. Les premiers tests fonctionnels ont donc été effectués :

- Le logiciel QEMU a été utilisé puisqu'il permet d'émuler une architecture et d'exécuter un ou plusieurs systèmes d'exploitation. Tester notre distribution sur une machine virtuelle nous permet d'identifier d'éventuelles erreurs avant que le logiciel soit installé sur la machine cible. Pour exécuter l'environnement virtuelle, la commande suivante a été utilisée :

```
$runqemu qemuarm
```

- Une fois l'environnement a été initialisé, le nom d'utilisateur et le mot de passe ont été saisis (utilisateur : ubss, mot de passe : ssi).

```
Poky (Yocto Project Reference Distro) 2.4.4 qemuarm tty
qemuarm login: ubss
Password: ssi
qemuarm:~$
```

Nous pouvons donc valider que le compte utilisateur fonction correctement.

- Nous pouvons aussi tester la connexion via SSH. L'adresse IP de la machine hôte est 192.168.10.2, l'adresse IP de la machine cible doit alors être configurée pour que les deux machines soient sur le même sous-réseau. Les commandes suivantes ont donc été utilisées :

```
Poky (Yocto Project Reference Distro) 2.4.4 qemuarm tty
qemuarm login: root
root@qemuarm:~#ifconfig eth0 192.168.10.1
```

Ceci a configuré notre machine cible avec l'adresse IP 192.168.10.1. La communication et le partage d'information sera donc plus facile, puisque les deux machines sont dans le même sous-réseau,

- Puis nous avons établi un pont de communication entre la machine cible et la machine hôte. Sur le terminal de la machine hôte, les commandes suivantes ont été utilisées :

```
$sudo ssh root@192.168.10.1
The authenticity of host '192.168.10.1 (192.168.10.1)' can't be established.
RSA key fingerprint is SHA256:1SIJ/499OrkZOS/VB+bs+TVRF9XStGhso9U96K0i2yE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.10.1' (RSA) to the list of known hosts.
root@qemuarm:~#
```

La connexion via SSH a été validée et la commande à distance peut être ainsi réalisée.

## Test de l'image sur la machine cible

Afin de créer une image pour la machine cible il faut modifier le fichier "local.conf". La modification suivante a été réalisée en utilisant l'éditeur nano :

```
#MACHINE ?= "qemuarm"
MACHINE ??= "solidrun-imx6"
```

La compilation a été faite à nouveau, puis elle a été montée sur une mémoire Micro SD. Tout d'abord, nous avons inséré une mémoire micro SD (en format ext4) et nous avons vérifié l'étiquette qui lui a été attribuée par le système d'exploitation.

```
$lsblk
NAME          MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
mmcblk1       179:0    0    7,3G  0 disk
```

Puis sur un autre terminal, l'environnement "oe-init-build-env" a été initialisé. Les commandes suivantes ont été utilisées pour monter l'image dans notre mémoire :

```
$cd tmp/deploy/images/solidrun-imx6/
$zcat core-image-ubs-homeautomation-solidrun-imx6.wic.gz | sudo dd of=/dev/mmcblk1
bs=4M iflag=fullblock oflag=direct conv=fsync status=progress
```

L'image a été correctement décompressée et placée sur la mémoire MicroSd. Comme l'image créée, a son propre u-boot notre distribution est prête à être testée. Les tests qui ont été effectués étaient les suivants :

- Comme la carte hummingboard n'a pas d'écran pour travailler, la communication série a été utilisée. Premièrement, le câble série a été correctement configuré et connecté à la machine hôte, puis sur le terminal de la machine hôte nous avons lancé la ligne de commande suivante :

```
$sudo screen /dev/ttyUSB0 1152000
```

Cela nous a permis de créer un pont de communication machine hôte - machine cible.

- Un fois le système démarré, la configuration d'IP de la machine cible était nécessaire. Les commandes suivantes ont été utilisées :

```
Poky (Yocto Project Reference Distro) 2.4.4 qemuarm ttymxc0
solidrun-imx6 login: root
root@solidrun-imx6: ~#ifconfig eth0 192.168.10.1
Atheros 8035 ethernet 2188000.ethernet-1:00:attached PHY driver [
Atheros 8035 ethernet] (mii_bus:phy_addr=2188000.ethernet-1:00, irq=-1)
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link is not ready
root@solidrun-imx6:~# fec 2188000.ethernet eth0:
Link is Up - 100Mbps/Full - flow control rx/tx
IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
```

De cette façon, deux communications machine hôte – machine cible sont disponibles. Le premier via série et le deuxième via SSH. Cependant la communication série sera utilisée plus tard pour envoyer les données du capteur Si7021 tandis que le SSH nous permettra de contrôler la machine cible à distance.

## Création et compilation d'une application

### Création d'une application

Afin de mesurer l'humidité et la température de l'environnement, le capteur "Adafruit Si7021" a été employé. Le capteur utilise une communication I2C, quatre fils ont donc été utilisés SLA, SDA, Vcc et GND. Sur la machine cible nous avons fait le branchement de SLA-SLA, SDA-SDA, Vcc-3.3v (puisque le voltage d'opération est de 1.9V à 3.6V) et GND-GND.

Le programme de base a été obtenu sur le site <https://bit.ly/2T61t7S>, cependant certaines modifications ont été faites :

- Pour déterminer sur quel port de communication le capteur a été branché, nous pouvons utiliser les commandes suivantes sur le terminal de communication SSH :

```

root@solidrun - imx6 :~# i2cdetect 2
WARNING ! This program can confuse your I2C bus , cause data loss
and worse !
I will probe file / dev / i2c -2.
I will probe address range 0 x03 -0 x77 .
43Continue ? [ Y / n ] Y
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

```

Le port à utiliser sur le programme est donc “/dev/i2c-2”. La modification faite sur le fichier .C a été le suivant :

```
char *bus = "/dev/i2c-2";
```

- Pour déterminer sur quel port de communication série le capteur doit afficher l’information, nous avons utilisé les commandes suivantes sur le terminal de communication SSH :

```

root@solidrun - imx6 :~# dmesg | grep tty
kernel command line: root=PARTUUID=2024ca51-01
root wait rw console=ttymx0, 115200
2020000.serial:ttymx0 at MMIO 0x2020000 (irq=27,
base_baud=50000000) is a IMX console [ttymx0] enabled
21ec000.serial:ttymx2 at MMIO 0x21ec000 (irq=74,
base_baud=50000000) is a IMX
21f0000.serial:ttymx3 at MMIO 0x21f0000 (irq=75,
base_baud=50000000) is a IMX serial serial0: tty port ttymx3 registered

```

Le port série ttymx0 est le seul disponible pour effectuer une communication série. Les modifications qui ont été prises en compte ont été la définition du port série, la configuration de la vitesse de communication, ainsi que certaines configurations d’un programme trouvé en ligne <https://bit.ly/2EPUooW>. Les modifications faites sur le fichier .C ont été les suivantes :

```

fd = open("/dev/ttymx0", O_RDWR | O_NOCTTY | O_NDELAY);
struct termios SerialPortSettings;
tcgetattr(fd, &SerialPortSettings);
cfsetispeed(&SerialPortSettings, B115200);
cfsetospeed(&SerialPortSettings, B115200);
SerialPortSettings.c_cflag &= ~PARENB;
SerialPortSettings.c_cflag &= ~CSTOPB;
SerialPortSettings.c_cflag &= ~CSIZE;
SerialPortSettings.c_cflag |= CS8;
SerialPortSettings.c_cflag &= ~CRTSCTS;
SerialPortSettings.c_cflag |= CREAD | CLOCAL;
SerialPortSettings.c_iflag &= ~(IXON | IXOFF | IXANY);
SerialPortSettings.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG);
SerialPortSettings.c_oflag &= ~OPOST;

```

- Le programme de base affiche les données d’humidité et de température qu’une seule fois. Cependant il est nécessaire que l’information soit affichée en permanence, c’est pourquoi une boucle a été utilisée pour faire défiler le processus d’acquisition des informations du capteur et d’affichage des informations.

- L'information obtenue par les capteurs est stockée dans une variable numérique, mais pour être transmise en série, ces valeurs doivent être transformées en un format de caractère. Les lignes suivantes ont été ajoutées sur le fichier .C :

```
sprintf(humi, "Humidity : %.2f ", humidity);
sprintf(ct, "Temp : %.2f \n\r ", cTemp);
```

- Finalement, les données sont envoyées par le port série. Une commande a été utilisée pour indiquer le port où les données seront envoyées, la taille du message et le message. Les modifications suivantes ont été ajoutées sur le fichier .C :

```
bytes_written = write(fd, humi, sizeof(humi));
bytes_written = write(fd, ct, sizeof(ct));
```

Un fois le programme soit correctement complété, le système pourra donc lire les données (température et humidité) du capteur Si7021 à intervalle de temps régulier et d'afficher le résultat de la lecture sur le port série UART.

## Compilation d'une application

Une compilation spéciale a été faite, puisqu'elle doit inclure toutes les fonctionnalités de notre machine cible, nous avons donc utilisé l'outil SDK que Yocto incorpore. L'outil SDK est un standard qui comprend les éléments suivants :

- Chaîne d'outils de développement croisé.
- Bibliothèques en-têtes et symboles.
- Script de configuration de l'environnement.

Avec le SDK nous pouvons développer et tester indépendamment le code destiné à fonctionner sur une machine cible. Nous partons de la base que nous avons déjà initialisé l'environnement (source oe-init-build-env) et nous sommes sur le dossier /poky/build :

- Puis nous avons utilisé la ligne de commande suivante pour créer le SDK :

```
bitbake meta-toolchain
```

- Pour installer correctement le SDK, nous avons exécuté le fichier.sh et nous avons suivi les instructions :

```
$ cd tmp/deploy/sdk
$ sudo ./poky-glibc-x86_64-meta-toolchain-cortexa9hf-neon-toolchain -2.4.4.sh
Poky (Yocto Project Reference Distro) SDK installer version 2.4.4
=====
Enter target directory for SDK (default: /opt/poky/2.4.4):
The directory "/opt/poky/2.4.4" already contains a SDK for this architecture.
If you continue, existing files will be overwritten! Proceed[y/N]? y
Extracting SDK.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source
the environment setup script e.g.
$ . /opt/poky/2.4.4/environment-setup-armv5e-poky-linux-gnueabi
$ . /opt/poky/2.4.4/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
```

- Le SDK a été correctement installé. Cependant pour compiler notre application il faut initialiser l'environnement "environment-setup-cortexa9hf-neon-poky-linux-gnueabi". Dans un autre terminal, nous avons occupé les commandes suivantes afin d'initialiser et compiler notre application :

```
$cd /opt/poky/2.4.4
source environment-setup-cortexa9hf-neon-poky-linux-gnueabi
cd /home/...[Adresse ou il se trouve le fichier a compiler]..../
$CC -o HTS HTSERIAL.c
```

Notre application HTSERIAL.c a été correctement compilé. Pour être testé sur la machine cible, il est nécessaire d'envoyer le document HTS. Le système de transfert de fichiers utilisé était la communication SSH et la commande 'copie'. Les étapes à suivre pour envoyer et tester l'application sont les suivantes :

- Afin d'envoyer l'application, nous avons défini le fichier source et le dossier du destine. Sur un nouveau terminal la commande suivante a donc été utilisée :

```
$sudo scp -r -p HTS root@192.168.10.1:/home/ubs/.
```

- Sur la terminal SSH de notre machine cible, nous avons exécuté l'application de la façon suivante :

```
root@solidrun - imx6 :~# cd /  
root@solidrun - imx6 :/# cd /home/ubs  
root@solidrun - imx6 :/home/ubs# ./HTS
```

Immédiatement sur le terminal série, il commencera à afficher des informations sur l'humidité [Humidity] et la température [Temp] de l'environnement.

## Création et initialisation d'un service

L'application peut être exécuté au démarrage du système, puisque notre distribution utilise le Systemd. La structure du service est simple, dans une première partie nous devons décrire le service, puis indique la tâche à exécuter, finalement nous devons indiquer le "target" où le service sera actif. Le fichier.service réalisé est le suivant :

```
[Unit]  
Description=HTS- Temperature et Humidite - Agent  
  
[Service]  
Type=idle  
ExecStart=/usr/libexec/HTS  
  
[Install]  
WantedBy=multi-user.target
```

Le fichier service "HTS-service.service" et le fichier "HTS" doivent être placés aux bons endroits pour leur exécution. Les étapes suivantes ont été suivies pour activer un service dans notre distribution :

- Le fichier .service a été placé sur le dossier du Systemd. L'application a été placé sur le dossier /usr/libexec. Les suivantes commandes ont donc été utilisées :

```
$sudo scp -r -p HTS-service.service root@192.168.10.1:/etc/systemd/system/.  
$sudo scp -r -p HTS root@192.168.10.1:/usr/libexec/.
```

---

**Note :** Sur le dossier /usr/libexec sont placés les binaires internes qui ne sont pas destinés à être exécutés directement par les utilisateurs ou les scripts shell.

---

- Sur la terminal SSH de notre machine hôte, nous avons exécuté le service de la façon suivante :

```
root@solidrun - imx6 :~# systemctl enable HTS-service.service.service
```

Ceci exécutera notre service à chaque fois que notre système démarre. Cependant si à tout moment nous voulons arrêter le service, la commande suivante peut être utilisée :

```
Poky (Yocto Project Reference Distro) 2.4.4 qemuarm tty
qemuarm login: root
root@solidrun - imx6 :~# systemctl stop HTS-service.service.service
```

**Note :** Comme le service s'exécute lorsque le système d'exploitation a été initialisé, il peut être difficile de placer les lignes de commande.

## Démarrage d'une distribution Linux via Emmc

La mémoire emmc est un endroit de stockage de la carte hummingboard, dans laquelle le système d'exploitation peut être introduit dans le but de ne pas dépendre d'une mémoire Micro SD. Cependant pour que le système d'exploitation puisse être installé dans la mémoire emmc, un autre système d'exploitation est nécessaire, puisqu'il nous permet de gérer les éléments matériels de l'architecture sur laquelle il est hébergé. Les étapes suivantes ont été utilisées pour sauvegarder notre distribution dans la mémoire emmc :

- Dans une mémoire USB l'image "core-image-ubs-homeautomation-solidrun-imx6.wic.gz" a été sauvegardée. Puis la mémoire a été insérée dans la machine cible. Afin d'accéder à tout l'information de la mémoire USB nous avons exécuté la commande suivante :

```
root@solidrun - imx6 :~# mount /dev/sda /mnt/
```

- Avant de monter l'image sur une partition, nous avons vérifié le nom de la partition. La commande suivante a été utilisée :

```
root@solidrun - imx6 :~# fdisk -l
```

- Dans la liste qui est affichée, nous pouvons observer que la partition mmcblk2 est la plus grande et c'est là que notre système d'exploitation peut fonctionner. Une fois le nom de la partition validé, l'image était montée sur cette partition.

```
root@solidrun - imx6 :~# cd /mnt
root@solidrun - imx6 :/mnt# zcat core-image-ubs-homeautomation-solidrun-imx6
.wic.gz | sudo dd of=/dev/mmcblk2
167936+0 records in
167936+0 records out
```

- Pour que le système puisse démarrer avec le mode emmc ou le mode Micro Sd, les jumpers doivent être placés comme suit :
  - Emmc : 3 jumpers seront utilisés : (1+2), (3+4), (7+8)
  - MicroSD : 2 jumpers seront utilisés : (3+4), (5+6)

Lors de l'initialisation en mode emmc, un message d'erreur Panic Kernel apparaît. Nous devons donc revenir à la configuration Micro SD et exécuter les commandes suivantes :

```
root@solidrun - imx6 :~# mount -t ext4 /dev/mmcblk2p1 /mnt/
root@solidrun - imx6 :~# cd /mnt
root@solidrun - imx6 :/mnt# cd boot/extlinux
root@solidrun - imx6 :/mnt/boot/extlinux# vi extlinux.conf
```

Cela nous permet de monter la partition mmcblk2p1 où notre distribution a été sauvegardée, avec le but de réaliser une modification au document extlinux.conf en utilisant l'éditeur Vi. Sur l'éditeur nous pouvons modifier le fichier en utilisant la touche 'i'. Nous devons modifier le fichier comme suit :

```
#Generic Distro Configuration file generated by OpenEmbedded
LABEL Poky (Yocto Project Reference Distro)
  KERNEL ../zImage
  FDTDIR ../
  APPEND root=/dev/mmcblk2p1 rootwait rw console=${console}
```

Pour finaliser le mode d'édition nous devons appuyer la touche 'esc' et puis les touches 'w' et 'q' pour enregistrer et quitter.

La configuration des 'Jumpers' doit être remplacée en mode emmc et le système peut être démarré sur la mémoire interne de la machine cible. Les étapes précédentes (2.5 et 2.6) doivent être répétées, car tout ce qui a été effectué a été stocké dans la mémoire Micro SD.

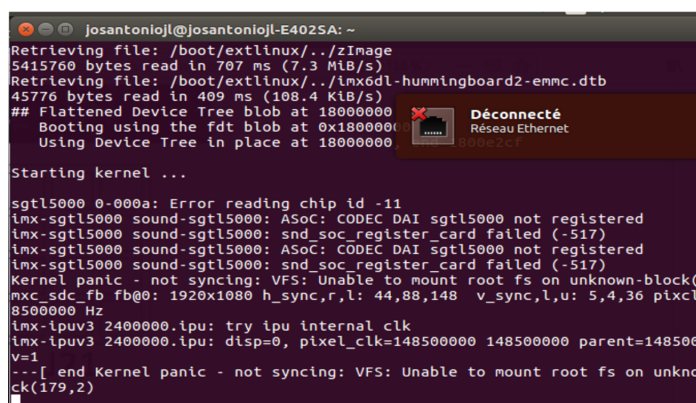


## Resultats

Comme toutes les étapes ont été correctement exécutées (2.1-2.6), le système fonctionne comme suit : <https://photos.app.goo.gl/7eKJKSoa1xxFsHw78>

Pour arrêter le service, le process suivante doit être effectuée : <https://photos.app.goo.gl/MLTC3iRKNcBNKVth9>

Lorsque l'image est enregistrée dans la mémoire emmc et que la configuration des "Jumpers" est effectuée, le message d'erreur suivant s'affiche :



```

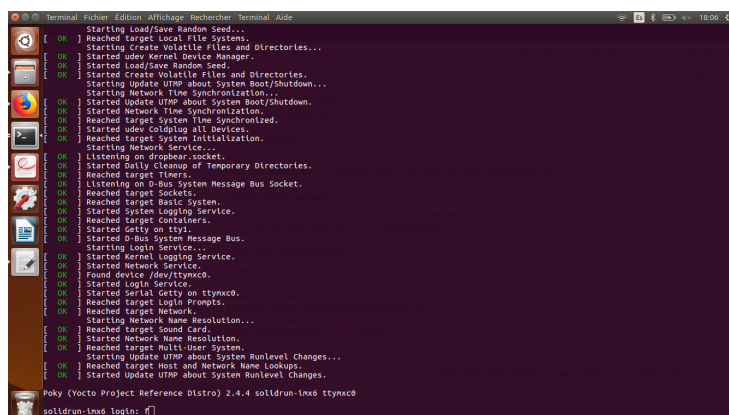
josantoniojl@josantoniojl-E4025A: ~
Retrieving file: /boot/extlinux/./zImage
5415760 bytes read in 707 ms (7.3 MiB/s)
Retrieving file: /boot/extlinux/./imx6dl-hummingboard2-emmc.dtb
45776 bytes read in 409 ms (108.4 KiB/s)
## Flattened Device Tree blob at 18000000
Booting using the fdt blob at 0x18000000
Using Device Tree in place at 18000000

Starting kernel ...

sgtl5000 0-000a: Error reading chip id -11
imx-sgtl5000 sound-sgtl5000: ASoC: CODEC DAI sgtl5000 not registered
imx-sgtl5000 sound-sgtl5000: snd_soc_register_card failed (-517)
imx-sgtl5000 sound-sgtl5000: ASoC: CODEC DAI sgtl5000 not registered
imx-sgtl5000 sound-sgtl5000: snd_soc_register_card failed (-517)
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(
mxs_sdc_fb fb@0: 1920x1080 h_sync,r,l: 44,88,148 v_sync,l,u: 5,4,36 pixel
8500000 Hz
imx-lpuv3 2400000.lpu: try ipu internal clk
imx-lpuv3 2400000.lpu: disp=0, pixel_clk=148500000 148500000 parent=148500
v=1
---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknow
ck(179,2)

```

Pour corriger l'erreur, il est nécessaire de changer la position des "Jumpers" et d'effectuer les changements décrits précédemment (2.7). Le système se chargera complètement depuis la mémoire emmc sans avoir besoin de la carte micro SD :



```

[ OK ] Reached target Local File Systems.
Starting Load/Save Random Seed...
[ OK ] Started udev Kernel Device Manager.
Starting Load/Save Random Seed.
[ OK ] Started Create Volatile Files and Directories.
Starting Update UTMP about System Boot/Shutdown...
[ OK ] Started Network Time Synchronization.
Starting Update UTMP about System Boot/Shutdown...
[ OK ] Started Network Time Synchronization.
Reached target System Time Synchronized.
Starting udev Coldplug all devices.
[ OK ] Reached target System Initialization.
Starting Network Service...
[ OK ] Listening on dropbear socket.
Starting Daily Cleanup of Temporary Directories.
[ OK ] Reached target Timers.
Listening on D-Bus System Message Bus Socket.
[ OK ] Reached target Sockets.
Reached target Basic System.
[ OK ] Started System Logging Service.
Reached target Containers.
[ OK ] Started Getty on tty1.
Starting D-Bus System Message Bus.
[ OK ] Started Login Service...
Starting Kernel Logging Service.
[ OK ] Started Network Service.
Found device /dev/ttnymcd.
[ OK ] Started Login Service.
Starting Serial Getty on ttnymcd.
[ OK ] Reached target Login Prompts.
Reached target Network.
Starting Network Name Resolution...
[ OK ] Reached target Sound Card.
Starting Network Name Resolution.
[ OK ] Reached target Multi-User System.
Starting Update UTMP about System Runlevel Changes...
[ OK ] Reached target Host and Network Name Lookups.
[ OK ] Started Update UTMP about System Runlevel Changes.

Poky (Yocto Project Reference Distro) 2.4.4 solidrun-imx6 ttnymcd
solidrun-imx6 login: #

```



---

# Conclusions

---

La réalisation de ce projet peut être comparé au projet SESI, puisque le niveau d'effort a aussi été élevé. Si bien, la création de l'image et la recette n'a été pas difficile, tous les sujets autour le projet l'ont été. Surtout parce que c'était la première fois dans ma vie que j'ai utilisé Linux. Quelques exemples des difficultés rencontrées étaient quelques commandes pour configurer l'adresse IP de la machine cible et la machine hôte, la connexion SSH, l'envoi des documents via SSH, l'édition des documents en mode vi et monter des mémoires externes.

Un autre exemple qui a également pris du temps a été la communication I2C et série sur la carte hummingboard, puisque je n'avais pas programmé ce type de communication dans un environnement Linux et en utilisant du code C.

Le principal défi du développement du projet a été de sauvegarder l'image de notre distribution dans la mémoire emmc. Si bien la procédure n'était pas difficile, certains détails doivent être considérés comme la configuration des 'jumpers'. Parce que si la configuration n'est pas bien faite l'OS jamais va démarrer via l'emmc, après de nombreuses essayes j'ai trouvé la légende suivante pendant le démarrage : "la partition emmc ne pouvait pas être initialisée", ce message m'a permis de comprendre l'erreur possible et dans la documentation de la carte, j'ai trouvé la configuration de 'Jumpers' nécessaires pour démarrer le système comme nous souhaitons. Par conséquent, une conclusion est de toujours avoir les manuels et fiches techniques des systèmes, puisque la source d'information est fiable et parceque en général ils contiennent toutes les informations nécessaires pour développer nos projets.

Si bien j'ai réussi de stocker l'image dans l'emmc, je n'avais pas le temps d'effectuer la détection de température et d'humidité, comme je l'avais fait dans la micro SD. L'emmc était la dernière exigence que j'avais fait et je n'ai eu plus le temps de la perfectionner. Certaines améliorations peuvent être apportées dans le travail, par exemple la modification de la recette, afin de compiler l'application, installer et démarrer le service et d'attribuer une adresse IP a la machine cible. De cette façon, notre image pourrait être reproduite pour un grand nombre de cartes de développement.

Finalement, tous les défis trouvés m'ont aidé à comprendre le fonctionnement de l'outil Yocto Project. J'ai appris de nouvelles commandes, et surtout je peux vous assurer que mes compétences dans l'environnement Linux ont considérablement progressé.



---

## Références

---

- de Yocto rocko mega manuel <https://www.yoctoproject.org/docs/2.4.3/mega-manual/mega-manual.html>
- de Yocto <https://wiki.solid-run.com/doku.php?id=products:imx6:software:os:yocto>
- du manuel de Bitbake <https://www.yoctoproject.org/docs/2.4.3/bitbake-user-manual/bitbake-user-manual.html>
- de Rocko <https://layers.openembedded.org/layerindex/branch/rocko/layers/>
- de Solid-Run <https://wiki.solid-run.com/doku.php?id=products:imx6:software:os:yocto>
- de Freescale <https://git.yoctoproject.org/cgit/cgit.cgi/meta-fsl-arm/about/>
- de la FSL Community BSP : <http://freescale.github.io/>
- de la machine Cible <https://wiki.solid-run.com/doku.php?id=products:imx6:hummingboard>