U-boot : Émulation de la carte ARM Versatile Express A9X4 via QEMU

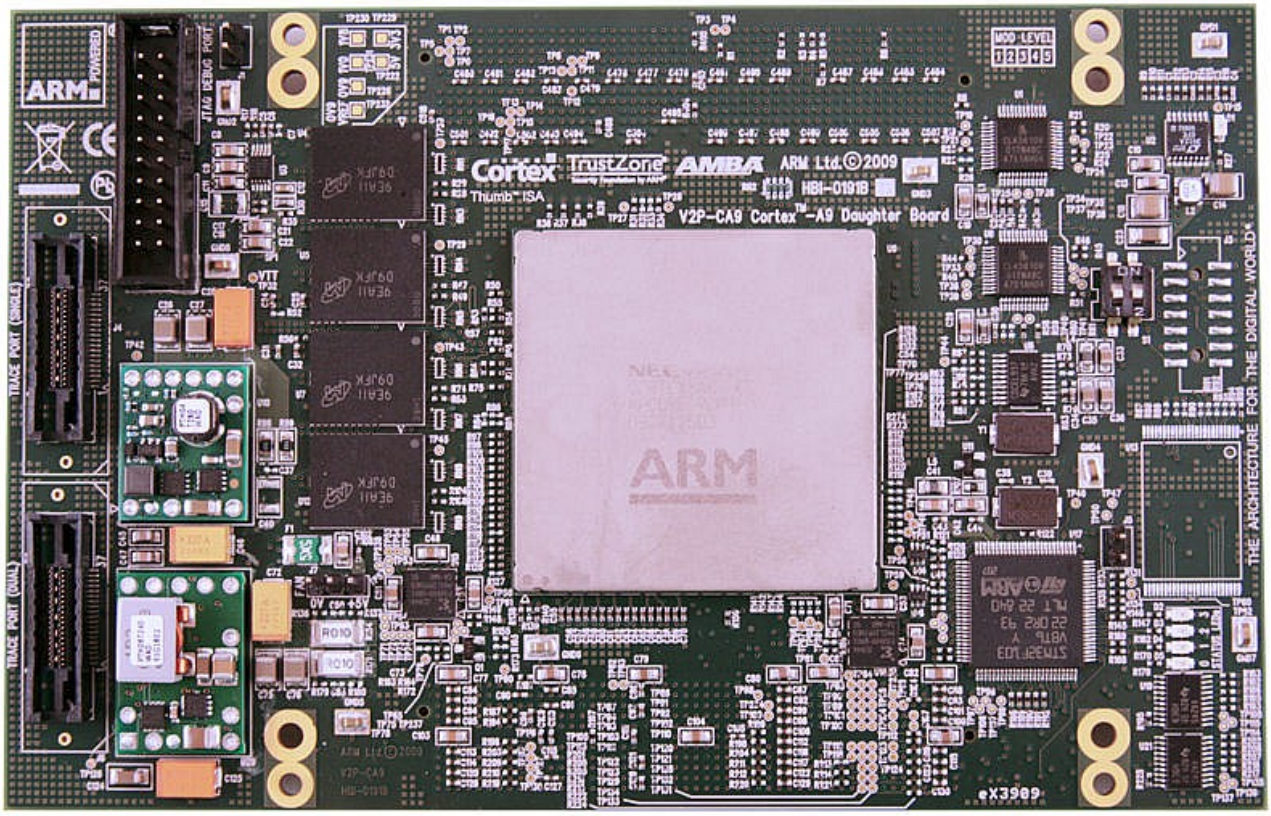

Figure 1: La carte de développement ARM versatile express[1]

.1. Introduction

Le chargeur de démarrage met principalement en place les bases pour que la carte puisse charger le noyau. le chargeur de démarrage ouvre la voie au noyau, allant jusqu'à le charger depuis un serveur distant en mémoire afin qu'il puisse être exécuté. Une fois qu'il est en mémoire, le chargeur de démarrage transmet l'exécution au noyau et se ferme. Le noyau a alors son propre processus d'amorçage (qui duplique ce que le chargeur de démarrage a fait dans certains cas) qui prépare le matériel, charge les pilotes et démarre ses processus pour gérer le système. Lorsqu'il est en cours d'exécution, le noyau cède le contrôle à un programme d'initialisation. Le noyau et init partagent ensuite la scène jusqu'à ce que vous arrêtiez le système.

Ce TP vise à tout rassembler, c'est-à-dire U-boot (un chargeur de démarrage), le noyau et le système de fichiers racine ramfs pour émuler une procédure de démarrage complète sur Qemu.

.2. Mise en place de la chaîne de cross-compilation

La différence entre un compilateur natif (gcc) et un compilateur croisé (par exemple arm-none-linux gnueabi-gcc) est que le compilateur natif s'exécute sur une architecture (par exemple x86_64) et produit des binaires pour la même architecture. Un compilateur croisé produit des binaires pour une architecture différente (dans notre cas ARMv7).

- Téléchargez la chaîne de cross compilation arm à partir de ce lien :

```
$wget https://releases.linaro.org/components/toolchain/binaries/6.3-2017.02/arm-linux-gnueabihf/gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf.tar.xz
```

- Décompresser l'archive téléchargée et mettez à jour la variable d'environnement PATH pour chaque session bash:

```
$cd ~

$tar -xvJf gcc-linaro-arm-linux-gnueabihf-4.8-2013.10_linux.tar.xz

Ajouter la ligne suivante à la fin du fichier ~/.bashrc :

export PATH=$PATH :/home/$USER/gcc-linaro-6.3.1-2017.02-x86_64_arm-linux-gnueabihf/bin' >> ~/.bashrc
```

- Tester la chaîne de cross-compilation en procédant à la compilation d'un programme **hello.c**

```c
#include <stdio.h>
int main()
{
        printf("La taille d'un entier long est %d",sizeof(long));
        return (sizeof(int));
}
```

- Ouvrez un nouveau terminal et procédez à la cross-compilation du fichier hello.c

```
◦ arm-linux-gnueabihf-gcc hello.c -o prog
◦ file prog # afin de voir le type de fichier
◦ ./prog # l'exécution est impossible dans votre pc
```

.3. Installation des packages nécessaires

installez le paquet qemu-system-arm. Dans ce TP et dans d'autres, nous utiliserons une carte ARM Vexpress Cortex A9 émulée par QEMU.

```
sudo apt-get install qemu-system-arm qemu-system-aarch64 tftp tftpd bison flex libssl-dev
```

.4. pré-configuration ,Cross compilation de u-boot et lancement de u-boot via qemu

.4.a) Téléchargement de u-boot

Dans un terminal procéder au téléchargement du code source de u-boot :

```
$git clone git://git.denx.de/u-boot.git

$cd u-boot/

$cd configs/

$ ls vexpress_* # on s'intéresse à la carte V2P-CA9x4
```

.4.b) Pré-configuration de u-boot selon la référence de la carte cible

Chargez la configuration par défaut pour la carte cible (versatile express A9X4) , c'est-à-dire **vexpress_ca9x4_defconfig** à l'aide de la commande suivante :

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- vexpress_ca9x4_defconfig
```

vérifier la création d'un fichier caché nommé .config qui contient la configuration de u-boot pour la carte cible, vous pouvez tout de même personnaliser cette configuration en faisant la commande :

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

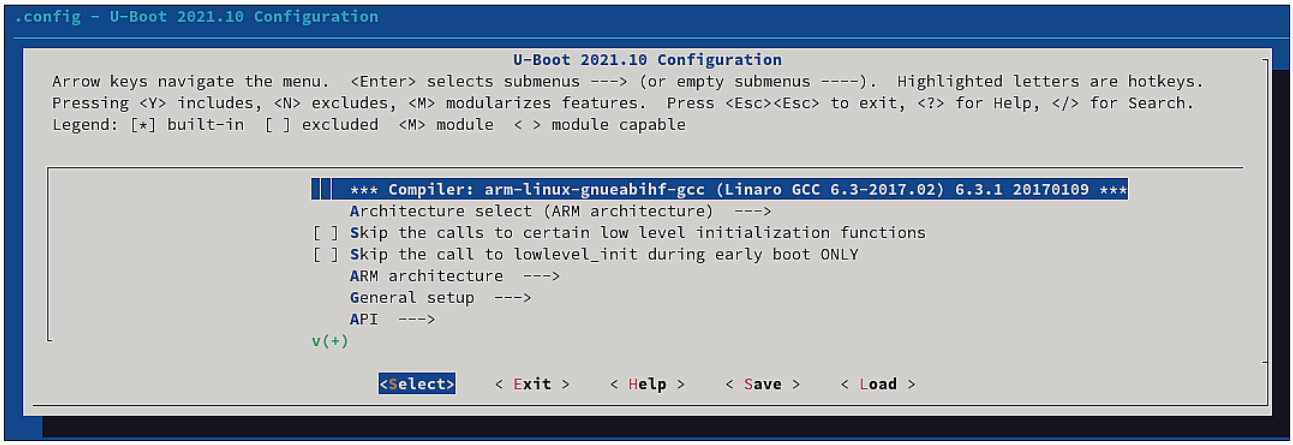

Figure 2: Configuration de u-boot pour la carte versatile express A9X4

.4.c) Cross-compilation de u-boot

cross-compiler U-boot à l'aide de cette commande :

```
ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- make -j`nproc`
```

```
[anouarchemek@toolbox u-boot]$ ls
Kbuild       Makefile    arch     config.mk  drivers   fs       post     u-boot            u-boot.cfg.configs  u-boot.sym
Kconfig      README      board    configs    dts       include  scripts  u-boot-nodtb.bin  u-boot.lds
Licenses     System.map  cmd      disk       env       lib      test     u-boot.bin        u-boot.map
MAINTAINERS  api         common   doc        examples  net      tools    u-boot.cfg        u-boot.srec
[anouarchemek@toolbox u-boot]$ file u-boot
u-boot: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, with debug_info, not stripped
[anouarchemek@toolbox u-boot]$
```

Figure 3: Résultat de cross-compilation de u-boot

.4.d) Lancement de u-boot via l'emulateur qemu

Lancer l'émulateur qemu de la carte cible via la commande suivante :

```
qemu-system-arm -M vexpress-a9 -m 512M -kernel u-boot -nographic
```

Les options utilisées avec la commande qemu-system-arm sont

-M : pour spécifier la machine à émuler, vous pouvez essayer la commande qemu-system-arm -M ? afin de vérifier est ce que l'émulateur qemu peut émuler la carte cible

-m : pour spécifier la taille de la mémoire ram nécessaire pour l'émulation de la carte, généralement cette valeur doit être la même qu'on trouve dans la carte réelle

-**kernel** : pour spécifier le noyau ou le boot-loader du noyau

-nographic : afin de lancer l'émulateur dedans le terminal actif et sans interface graphique.

```
[anouarchemek@toolbox u-boot]$ qemu-system-arm -M vexpress-a9 -m 512M -kernel u-boot -nographic
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument


U-Boot 2021.10-00815-gf200a4bcec (Oct 21 2021 - 22:11:21 +0100)

DRAM:  512 MiB
WARNING: Caches not enabled
Flash: 64 MiB
MMC:   mmci@5000: 0
Loading Environment from Flash... *** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   eth0: ethernet@3,02000000
Hit any key to stop autoboot:  0
MMC Device 1 not found
```

Figure 4: Lancement de u-boot (partie 1)

```
TFTP error: trying to overwrite reserved memory...
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC 52:54:00:12:34:56
BOOTP broadcast 1
DHCP client bound to address 10.0.2.15 (2 ms)
Using ethernet@3,02000000 device
TFTP from server 10.0.2.2; our IP address is 10.0.2.15
Filename 'boot.scr.uimg'.
Load address: 0x60100000
Loading: *
TFTP error: 'Access violation' (2)
Not retrying...
smc911x: MAC 52:54:00:12:34:56
cp - memory copy

Usage:
cp [.b, .w, .l, .q] source target count
Wrong Image Format for bootm command
ERROR: can't get kernel image!
=>
```

Figure 5: Lancement de u-boot (partie 2)

.5. Patch,configuration et Cross compilation du noyau Linux

.5.a) Téléchargement du noyau Linux

Dans un terminal, procéder au téléchargement du code source du noyau Linux tout en pointant sur la branche stable v4.8

```
$ git clone --branch v4.8 --depth=1 https://github.com/torvalds/linux.git
```

.5.b) Pré-configuration du noyau linux pour versatile express A9x4

Charger la configuration du noyau par défaut pour la carte cible (versatile express A9X4) , c'est-à-dire **vexpress_defconfig** à l'aide de la commande suivante :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- vexpress_defconfig
```

Afin de vérifier l'existence de cette pré-configuration de la carte cible, on accède à linux/arch/arm/configs/ , dans un tel emplacement on trouve des fichiers de configurations pour plusieurs cartes embarqués ARM

.5.c) Reconfiguration du noyau Linux pour u-boot

On doit modifier quelques paramètre dedans la pré-configuration avec la commande :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

Suivez les étapes suivantes du menu textuel de configuration :

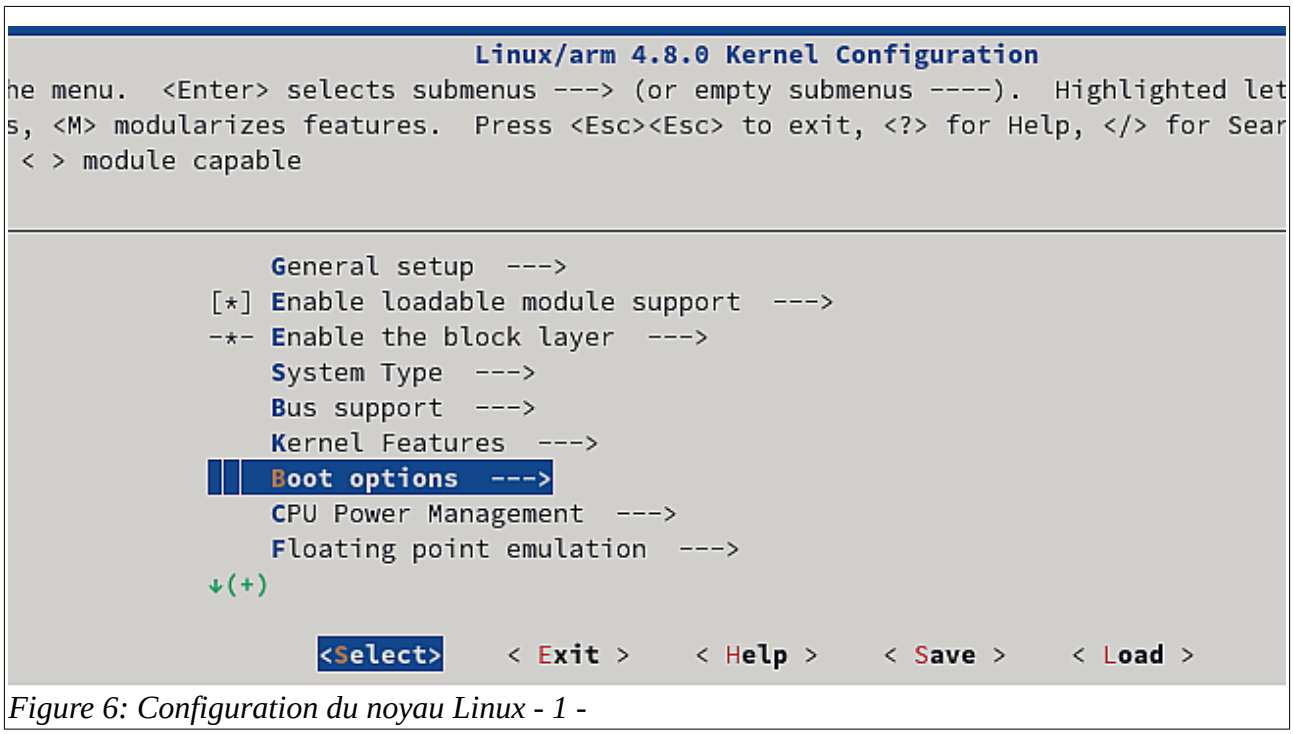

Figure 6: Configuration du noyau Linux - 1 -

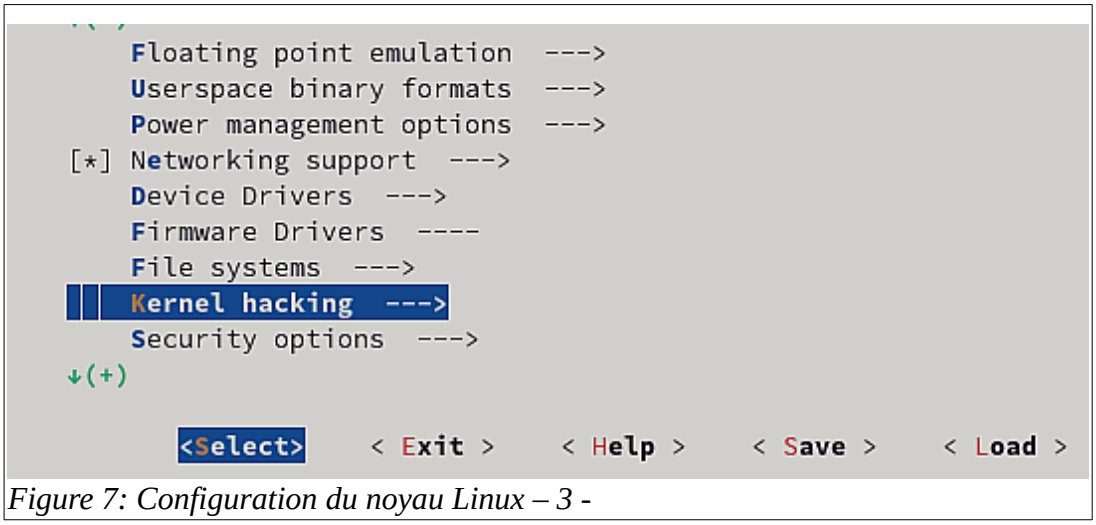

Figure 7: Configuration du noyau Linux – 3 -

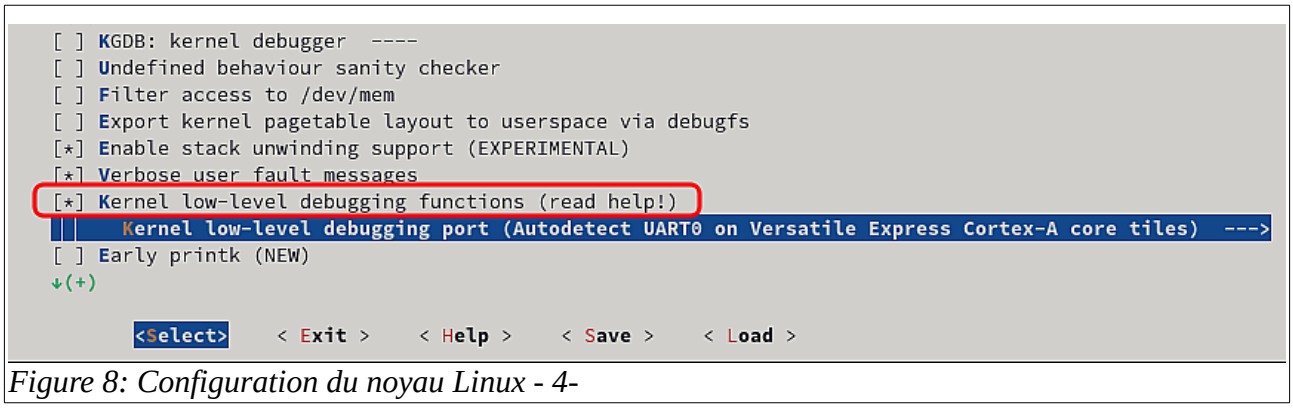

Figure 8: Configuration du noyau Linux - 4-

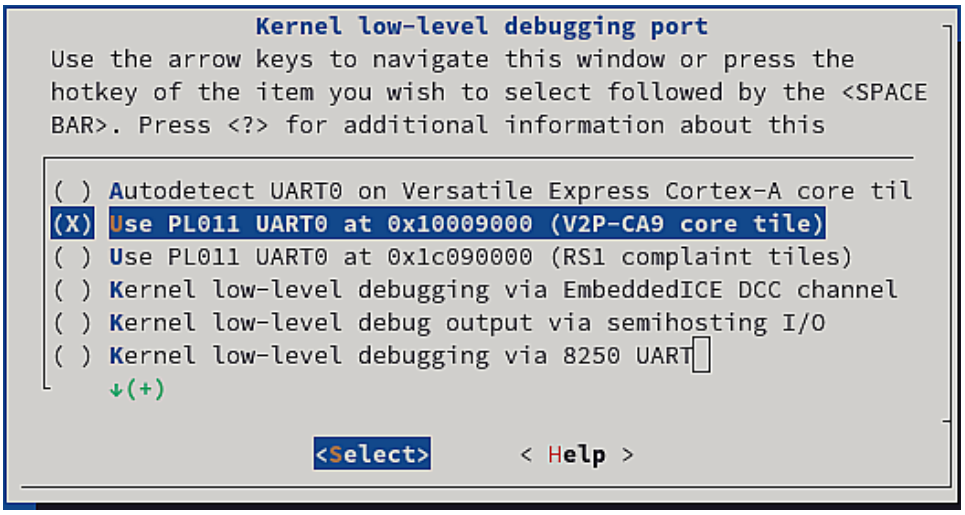

Figure 9: Configuration du noyau Linux - 5 -

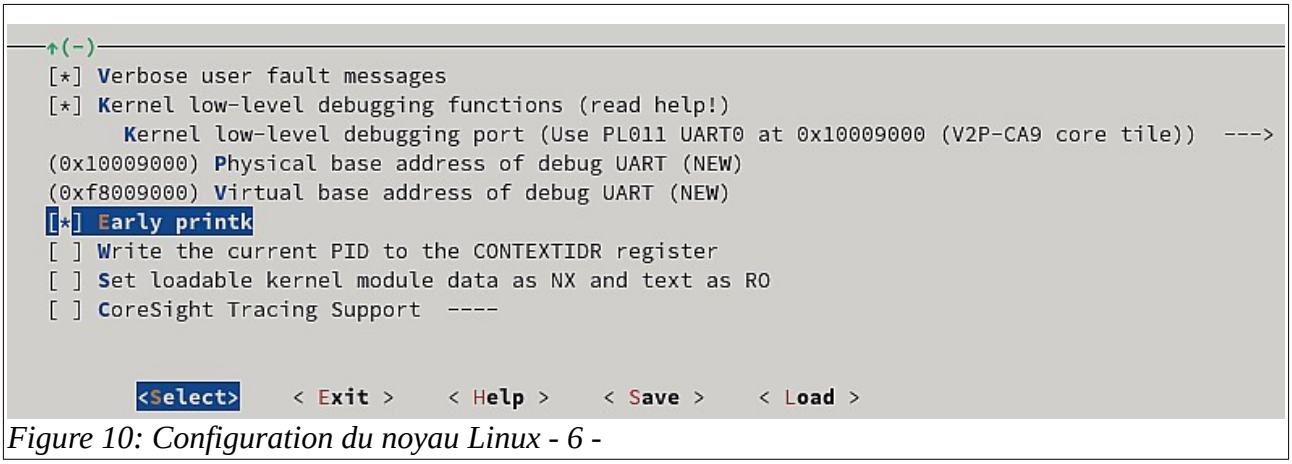

Figure 10: Configuration du noyau Linux - 6 -

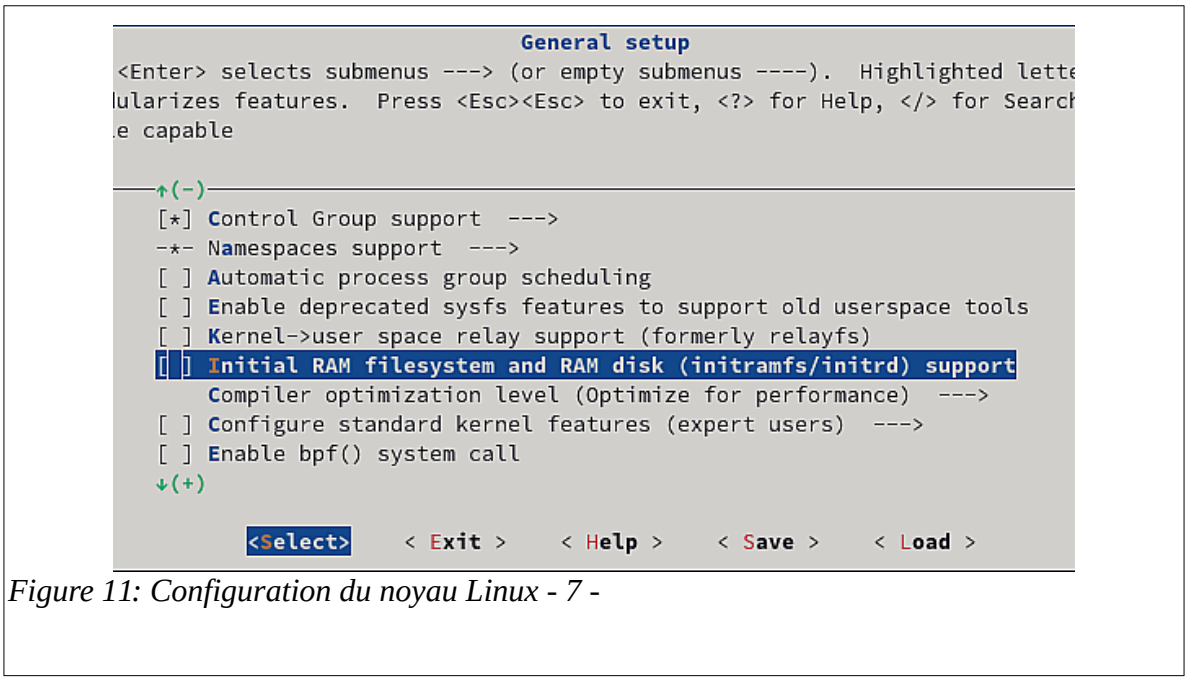

Figure 11: Configuration du noyau Linux - 7 -

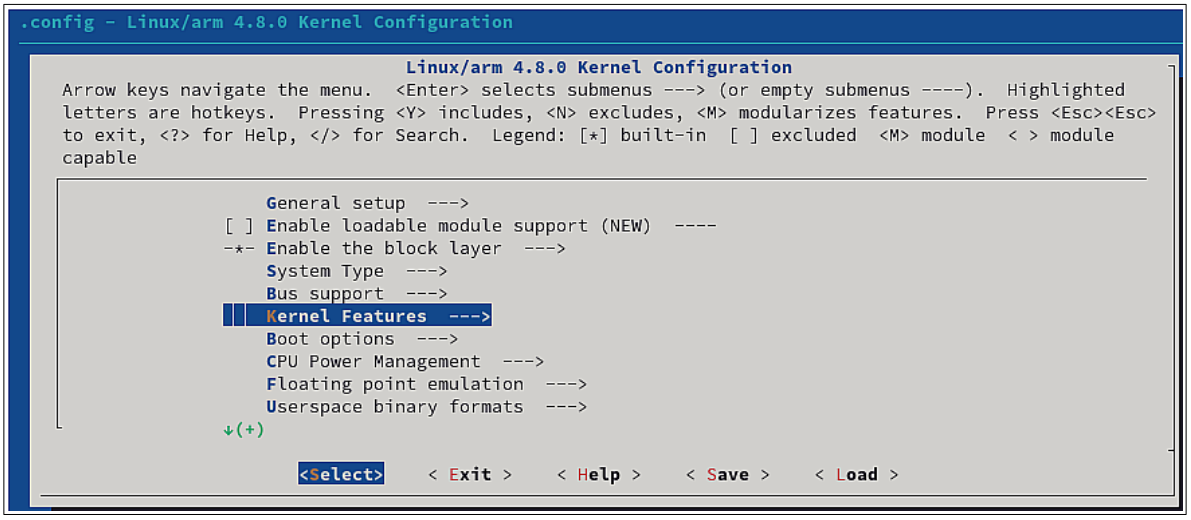

Figure 12: Configuration du noyau Linux - 9 -

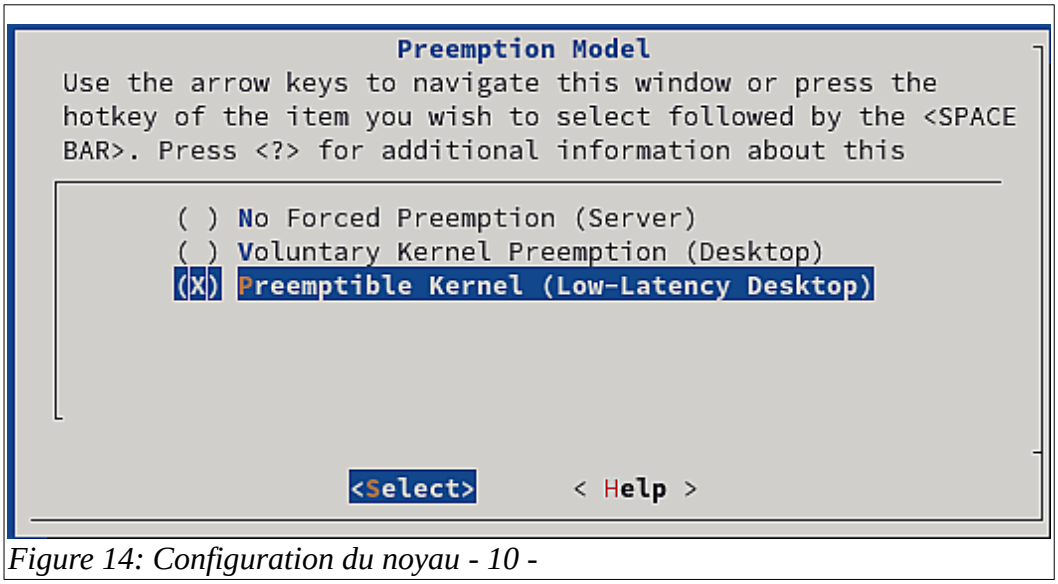

Figure 14: Configuration du noyau - 10 -

N’oubliez pas de sauvegarder cette nouvelle configuration et de quitter

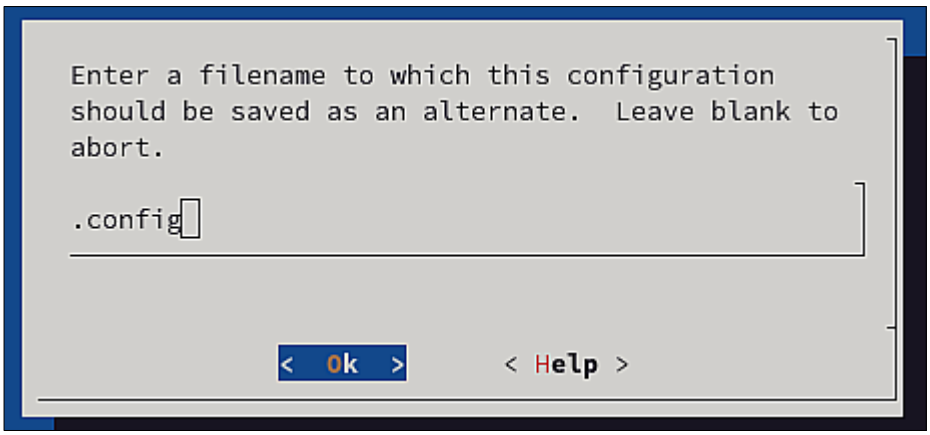

Figure 15: Sauvegarde de la nouvelle configuration du noyau Linux

.5.d) Application d'un patch rt au noyau linux

a) Définition et Création d'un fichier patch

Un **patch** est un petit document texte contenant quelques modifications entre deux versions différentes d'une arborescence source. Les patchs sont créés avec le programme **diff**. On utilisant la commande **diff** entre deux versions de code, on pourra obtenir un fichier de différence entre les deux versions qu'on nomme généralement un fichier patch.

```
[esprit@localhost ~]$ diff main.c main_updated.c > patch
```

Figure 16: Création d'un fichier patch via la commande diff

Maintenant, il est possible d'appliquer le fichier **patch** résultant sur le fichier **main.c** afin d'obtenir une nouvelle révision de ce même fichier (main.c)

```
[anouarchemek@toolbox test]$ ls
main.c  patch
[anouarchemek@toolbox test]$ cat patch | patch -p0 --dry-run
checking file main.c
[anouarchemek@toolbox test]$ cat patch | patch -p0
patching file main.c
[anouarchemek@toolbox test]$ cat main.c
#include <stdio.h>
#include <math.h>
int main(int argc,char **argv)
{
        printf("Bonjour\ncos(pi)=%f",cos(M_PI));
        return 0;
}
[anouarchemek@toolbox test]$ cat patch | patch -R -p0
patching file main.c
[anouarchemek@toolbox test]$ cat main.c
#include <stdio.h>
int main()
{
        printf("Bonjour\n");
        return 0;
}
[anouarchemek@toolbox test]$
```

Figure 17: Application d'un patch sur une révision du code source

b) Définition du patch linux-rt

Le patch PREEMPT_RT est disponible pour chaque version stable à long terme du noyau Linux principal depuis la version du noyau v2.6.11. La plupart des versions stables à long terme du noyau Linux ont un numéro de sous-version pair, exemple le noyau linux-4.8.

Afin de s'acquérir le patch linux-rt de la version linux-4.8, tapez la commande suivante :

```
$wget https://cdn.kernel.org/pub/linux/kernel/projects/rt/4.8/older/patch-4.8-rt1.patch.xz
```

c) Application du patch linux-rt sur le noyau linux-4.8 et reconfiguration du noyau

Le fichier compressé patch-4.8-rt1.patch.xz, que vous devenez de télécharger, doit être mis dans le même chemin du noyau linux, la figure suivante présente un aperçu des emplacements du fichier de patch compressé ainsi que du code source du noyau Linux :

```
[anouarchemek@toolbox 4iosys]$ ls -ld linux/ patch-4.8-rt1.patch.xz
drwxr-xr-x. 1 anouarchemek anouarchemek    964 Oct 28 22:52 linux/
-rw-r--r--. 1 anouarchemek anouarchemek 161976 Oct  6  2016 patch-4.8-rt1.patch.xz
[anouarchemek@toolbox 4iosys]$ ls linux/
COPYING        MAINTAINERS     System.map  drivers   ipc              modules.order  sound    vmlinux.o
CREDITS        Makefile        arch        firmware  kernel           net            tools
Documentation  Module.symvers  block       fs        lib              samples        usr
Kbuild         README          certs       include   localversion-rt  scripts        virt
Kconfig        REPORTING-BUGS  crypto      init      mm               security       vmlinux
[anouarchemek@toolbox 4iosys]$
```

Figure 18: Chemin des sources du noyau linux et du patch compressé

Accéder au code source du noyau linux/ et tapez les commande suivantes :

```
[anouarchemek@toolbox 4iosys]$ cd linux/
[anouarchemek@toolbox linux]$ xzcat ../patch-4.8-rt1.patch.xz | patch -p1 --dry-run
checking file Documentation/hwlat_detector.txt
checking file Documentation/sysrq.txt
checking file Documentation/trace/histograms.txt
checking file arch/Kconfig
checking file arch/arm/Kconfig
checking file arch/arm/include/asm/switch_to.h
checking file arch/arm/include/asm/thread_info.h
checking file sound/core/pcm_native.c
[anouarchemek@toolbox linux]$ xzcat ../patch-4.8-rt1.patch.xz | patch -p1
patching file Documentation/hwlat_detector.txt
patching file Documentation/sysrq.txt
patching file Documentation/trace/histograms.txt
patching file arch/Kconfig
patching file arch/arm/Kconfig
patching file arch/arm/include/asm/switch_to.h
patching file arch/arm/include/asm/thread_info.h
patching file arch/arm/kernel/asm-offsets.c
patching file arch/arm/kernel/entry-armv.S
```

Figure 19: Application du patch linux-rt sur le code source du noyau Linux

Une fois que nous obtenons une révision du noyau linux-rt via l'application du patch-4.8-rt1.patch.xz, n'oubliez pas de ré-configurer le noyau avec la commande suivante

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

et tachez de modifier le modèle de préemption du noyau afin qu'il devient Fully Preemptible Kernel (RT) au lieu de Preemptible Kernel (Low-Latency Desktop)

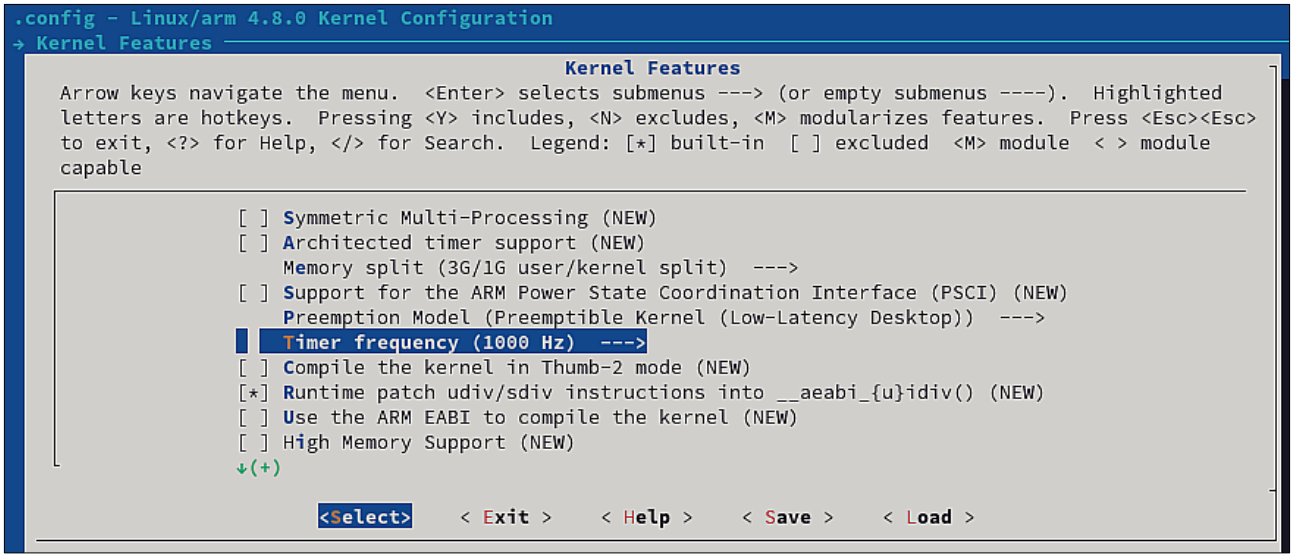

Figure 20: Configuration du noyau Linux - 11 -

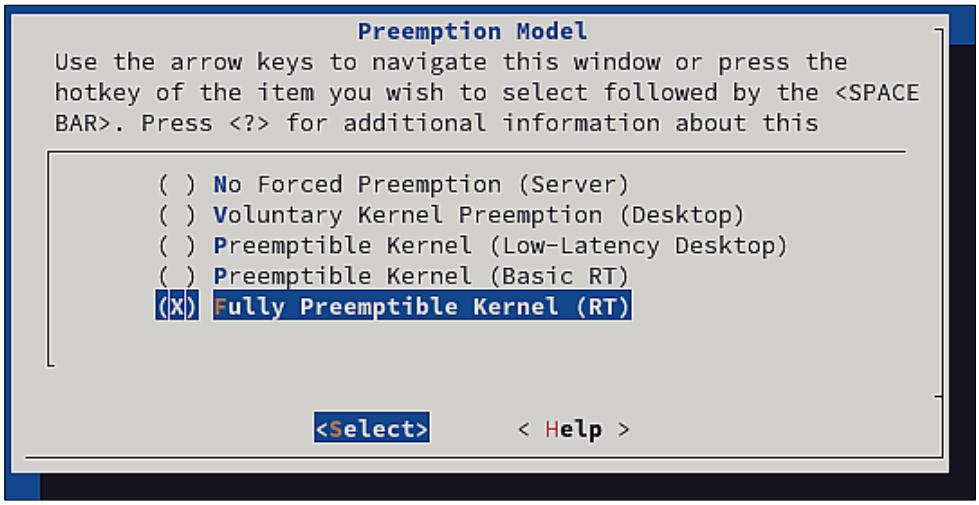

Figure 21: Configuration du noyau Linux - 12 -

N’oubliez pas de sauvegarder cette nouvelle configuration et de quitter

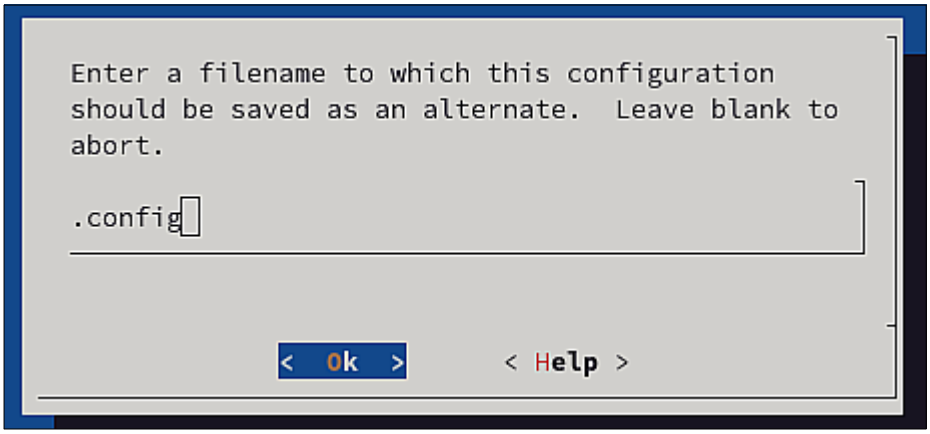

Figure 22: Sauvegarde de la nouvelle configuration du noyau Linux

.5.e) Cross-compilation du noyau Linux

Cross-compiler le noyau linux-4.8 via la commande suivante :

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- all -j`nproc`
```

Depuis le repertoire du noyau linux, accéder à **arch/arm/boot/** vous devrez trouver le noyau compilé et compressé nommé **zImage**

```
[anouarchemek@toolbox linux]$ cd arch/arm/boot/
[anouarchemek@toolbox boot]$ ls
Image  Makefile  bootp  compressed  dts  install.sh  zImage
[anouarchemek@toolbox boot]$ cd dts/
[anouarchemek@toolbox dts]$ ls *.dtb
vexpress-v2p-ca15-tc1.dtb  vexpress-v2p-ca15_a7.dtb  vexpress-v2p-ca5s.dtb  vexpress-v2p-ca9.dtb
[anouarchemek@toolbox dts]$
```

Figure 23: Résultat de cross-compilation du noyau Linux pour la carte ARM versatile express A9X4

.6. Chargement du noyau Linux via u-boot

.6.a) Configuration du service tftp[2]

TFTP (Trivial File Transfer Protocol ou protocole simplifié de transfert de fichiers) est un protocole simplifié de transfert de fichiers généralement utilisé en réseau local. Il utilise le protocole UDP de la couche de transport via le port 69, au contraire du FTP qui utilise le protocole TCP.

Sous Linux Ubuntu-18.04, l'usage de tftp pourra être géré via le daemon xinetd (eXtended InterNET Daemon) via une configuration située dans **/etc/xinetd.d/tftp**

- Installer les services tftp et xinetd via la commande suivante :

```
sudo apt-get update && sudo apt-get install xinetd tftpd tftp
```

- Configurer xinetd afin qu'il prend en charge le service tftp
    - Éditer le fichier **/etc/xinetd.d/tftp** et mettre dedans ce contenu:

```
service tftp
{
protocol = udp
port = 69
socket_type = dgram
wait = yes
user = nobody
server = /usr/sbin/in.tftpd
server_args = /tftpboot
disable = no
}
```

- Créer le dossier **/tftpboot** avec les commandes suivantes :

```
sudo mkdir /tftpboot/
sudo chmod -R 777 /tftpboot/
sudo chown -R nobody /tftpboot/
```

- Tester le service tftp avec les commandes suivantes :
  - $ sudo service xinetd restart
  - $ echo 'Ceci est un test' > /tftpboot/test
  - tftp 127.0.0.1

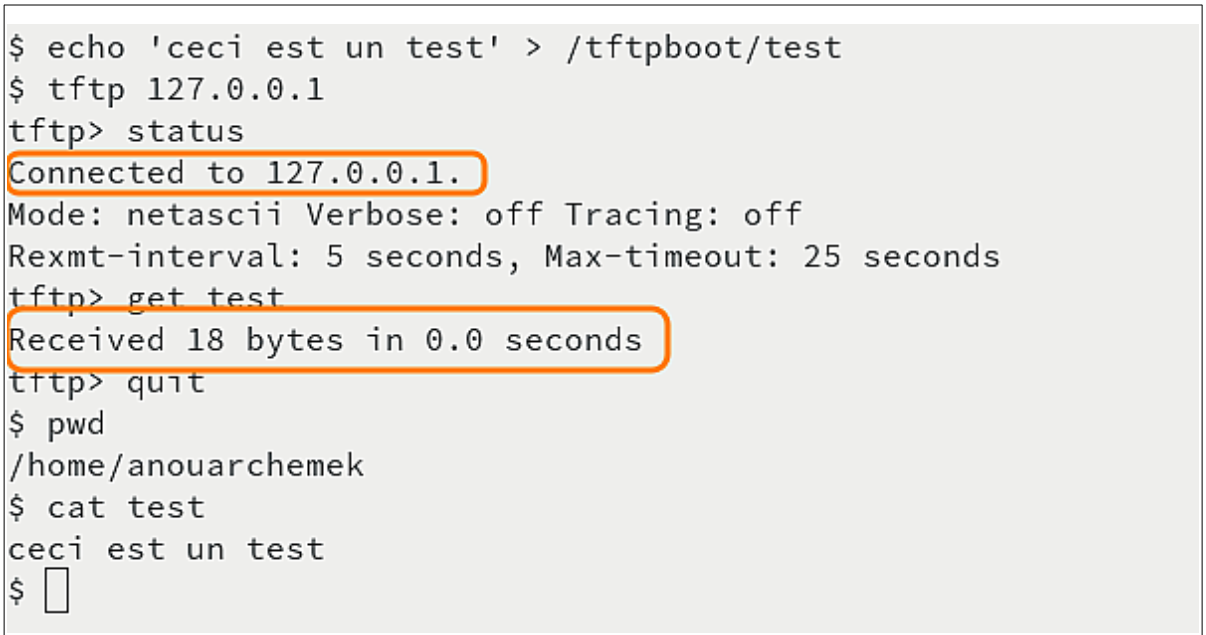

Figure 24: test de tftp en local

Une fois le serveur tftp fonctionne comme indiqué dans la figure 24, vous pouvez placer dans l'emplacement /tftpboot/ le noyau linux compressé ainsi que le le fichier dtb associé au noyau :

- cd linux/
- cp arch/arm/boot/zImage /tftpboot/
- cp arch/arm/boot/dts/vexpress-v2p-ca9.dtb /tftpboot/

.6.b) Configuration d'une interface réseau virtuel (tap1)

Afin de permettre le chargement du noyau via u-boot moyennant le service tftp, il faut avoir une interface réseau virtuelle entre le système d'exploitation Ubuntu-18.04 et

l'émulateur qemu. L'interface qu'on va créer est **tap1** qui utilisera un driver nommé **tun** et qui sera crée avec l'utilitaire **tunctl.**

Via Ubuntu-18.04, installer les dépendances manquantes :

- $ sudo apt-get install uml-utilities
- $ sudo modprobe tun
- sudo tunctl -t tap1
- sudo ifconfig tap1 192.168.1.1 netmask 255.255.255.0 up

```
anouarchemek@e584593c9cc0:~$ sudo ifconfig tap1 192.168.1.1 netmask 255.255.255.0 up
anouarchemek@e584593c9cc0:~$ ifconfig tap1
tap1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.1.1  netmask 255.255.255.0  broadcast 192.168.1.255
        ether 92:2a:7f:53:31:ae  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

anouarchemek@e584593c9cc0:~$
```

Figure 25: Création d'une interface virtuelle tap1

- Accéder au répertoire u-boot et relancer qemu-system-arm avec une prise en main du réseau via les commandes suivantes :
  - cd ~/u-boot/
  - sudo **qemu-system-arm** -**M** vexpress-a9 -**m** 512M -**kernel** u-boot -**net** nic -**net** tap,ifname=tap1,script=no,downscript=no -**nographic**

.6.c) Chargement et exécution du noyau linux via qemu

Une fois que u-boot est lancé avec la prise en main du réseau et afin de permettre le chargement du noyau linux via u-boot moyennant le service tftp, il faut paramétrer u-boot avec la bonne adresse ip (**ipaddr**)ainsi que l'adresse du serveur tftp (**serverip**) et le bon maque de sous réseau (**netmask**).

Le noyau lors de son chargement on pourra le paramétrer (voir ce lien)

U-boot va télécharger depuis le serveur tftp le noyau compressé (**arch/arm/boot/zImage**) ainsi que le le fichier **dtb (linux/arch/arm/boot/dts/vexpress-v2p-ca9.dtb)**

Il est à noter que DTB (Device tree blob / binary), est un ensemble de structures de données qui représente les composants matériels de la carte cible. U-Boot transmet la structure d'informations de la carte au noyau.

Dans la console de u-boot, lancer les commandes suivantes :

- setenv ipaddr 192.168.1.2
- setenv serverip 192.168.1.1
- setenv netmask 255.255.255.0
- setenv bootargs 'root=/dev/mmcblk0 console=ttyAMA0,38400n8'

```
Retry time exceeded; starting again
smc911x: MAC 52:54:00:12:34:56
cp - memory copy

Usage:
cp [.b, .w, .l, .q] source target count
Wrong Image Format for bootm command
ERROR: can't get kernel image!
=> setenv ipaddr 192.168.1.2
=> setenv serverip 192.168.1.1
=> setenv netmask 255.255.255.0
=> setenv bootargs 'root=/dev/mmcblk0 console=ttyAMA0,38400n8'
=>
```

Figure 26: paramétrage de u-boot afin de permettre le chargement du noyau Linux

Toutefois, il est possible via le prompt de u-boot de tester la connectivité avec la machine host (ubuntu-18.04)

```
=> ping 192.168.1.1
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC 52:54:00:12:34:56
Using ethernet@3,02000000 device
smc911x: MAC 52:54:00:12:34:56
host 192.168.1.1 is alive
=>
```

Figure 27: Test de la connectivité entre la machine cible et la machine hôte via u-boot

D'après, ce que nous avons mis comme arguments au noyau Linux, ce dernier permettra de le chargement du premier processus (sysVinit ou systemd) depuis une carte micro SD (root=/dev/mmcblk0)

Pour la carte ARM versatile express a9x4, Le chargement du noyau linux sera effectué dans la mémoire RAM, qu'on doit connaître son adresse de début et de fin, dans la

figure suivante (figure 28), on peut déduire pour une ram de 512Mo, l'adresse de début est **0x6000000** et de fin est **0x7FFFFFFF**

Memory Address Range	Size	Description
0x60000000-0x7FFFFFFF	512MB	RAM ⩽ 512MB
0x80000000-0x9FFFFFFF	512MB	RAM > 512MB
0x40000000-0x44000000	64MB	NOR FLASH 1
0x48000000-0x4a000000	64MB	NOR FLASH 2

Figure 28: mappage des adresses pour la mémoire RAM et NOR Flash (1 &2)

```
host 192.168.1.1 is alive
=> bdinfo
boot_params = 0x60002000
DRAM bank   = 0x00000000
-> start    = 0x60000000
-> size     = 0x20000000
DRAM bank   = 0x00000001
-> start    = 0x80000000
-> size     = 0x00000004
flashstart  = 0x40000000
flashsize   = 0x04000000
flashoffset = 0x00000000
baudrate    = 38400 bps
relocaddr   = 0x7ff6b000
```

Figure 29: visualisation des adresses pour le DRAM bank 0 et 1

Pour éviter tout chevauchement entre **u-boot** et le noyau **linux** lors du chargement, il est utile de connaître que u-boot va se charger à 8MB depuis le commencement de la RAM donc c'est forcement l'adresse **0x60800000** (voir fichier **u-boot.cfg**)

```
anouarchemek@3b3582999be4:~/fsl-community-bsp/TP2/u-boot$ ls
api      config.mk  dts       Kbuild       Makefile  System.map  u-boot.cfg          u-boot.srec
arch     configs    env       Kconfig      net       test        u-boot.cfg.configs  u-boot.sym
board    disk       examples  lib          post      tools       u-boot.lds
cmd      doc        fs        Licenses     README    u-boot      u-boot.map
common   drivers    include   MAINTAINERS  scripts   u-boot.bin  u-boot-nodtb.bin
anouarchemek@3b3582999be4:~/fsl-community-bsp/TP2/u-boot$ cat u-boot.cfg | grep CONFIG_SYS_TEXT_BASE
#define CONFIG_SYS_TEXT_BASE 0x60800000
anouarchemek@3b3582999be4:~/fsl-community-bsp/TP2/u-boot$
```

Figure 30: Adresse de chargement de u-boot dans la mémoire RAM

Figure 31: Vérification du contenu de la mémoire avec la commande md de u-boot

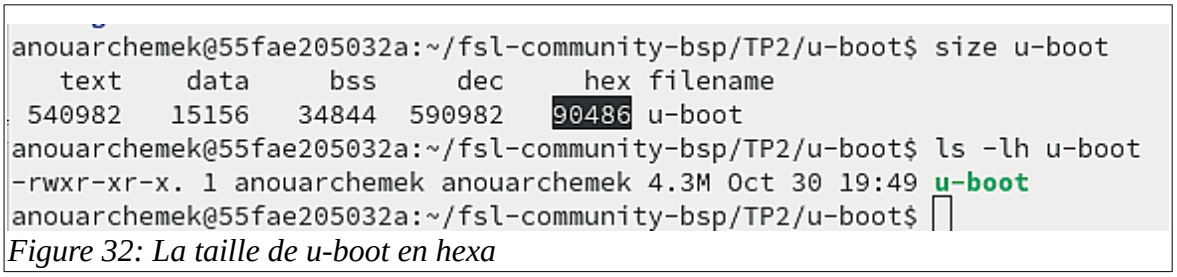

Figure 32: La taille de u-boot en hexa

On pourra voir dans la figure suivante (figure 32) que l'adresse **0x60900000** (situé plus loin dans la mémoire RAM) est vide et à partir de la on pourra charger le noyau sans soucis de chevauchement avec **u-boot**.

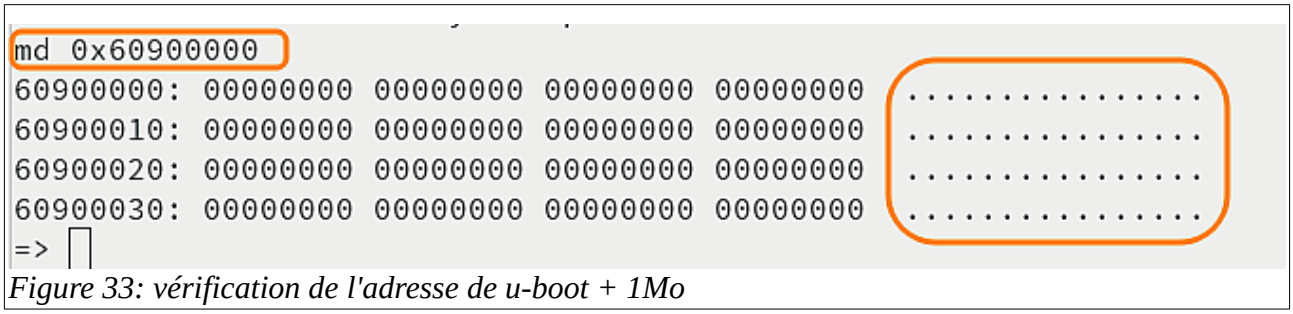

Figure 33: vérification de l'adresse de u-boot + 1Mo

Il ne reste que de charger le noyau via tftp et son amorçage à l'adresse souhaité qui est **0x60900000**

```
=> tftp 0x60a00000 zImage
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC 52:54:00:12:34:56
Using ethernet@3,02000000 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'zImage'.
Load address: 0x60a00000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##########################################################
         1.2 MiB/s
done
Bytes transferred = 3611432 (371b28 hex)
smc911x: MAC 52:54:00:12:34:56
=>
```

Figure 34: Chargement du noyau Linux à l'adresse 0x60a00000

N'oublions pas aussi le chargement du fichier **vexpress-v2p-ca9.dtb** qui contient les données structurés des périphériques de la carte ARM versatile express axc9 que u-boot transmuterais au noyau Linux.

L'adresse à laquelle le fichier dtb sera chargé est **0x60010000** . Cette adresse est au bas de la mémoire ram avec un offset de 64 KB

```
=> tftp 0x60010000 vexpress-v2p-ca9.dtb
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC 52:54:00:12:34:56
Using ethernet@3,02000000 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'vexpress-v2p-ca9.dtb'.
Load address: 0x60010000
Loading: ###
         410.2 KiB/s
done
Bytes transferred = 14708 (3974 hex)
smc911x: MAC 52:54:00:12:34:56
=>
```

Figure 35: Chargement de vexpress-v2p-ca9.dtb en RAM

La dernière étape consiste à amorcer le noyau avec la commande **bootz :**

```
bootz 0x60900000 – 0x60010000
```

```
=> bootz 0x60900000 - 0x60010000
Kernel image @ 0x60900000 [ 0x000000 - 0x371b28 ]
## Flattened Device Tree blob at 60010000
   Booting using the fdt blob at 0x60010000
   Loading Device Tree to 7fb20000, end 7fb26973 ... OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0x0
Linux version 4.8.0-rt1+ (anouarchemek@ff178362761d) (gcc version 7.5.0 (Ubuntu/Linaro 7.5.0-3ubuntu1~18.04)
2:29:46 UTC 2021
CPU: ARMv7 Processor [410fc090] revision 0 (ARMv7), cr=10c5387d
```

Figure 36: Décompression et lancement du noyau Linux

Le chargement du noyau va se terminer avec un echec parqce que le rootfs est inexistant, on dispose pas d'un support de boot sous forme de carte micro SD comme ça était spécifié aux bootargs du noyau linux via u-boot

```
VFS: Cannot open root device "mmcblk0" or unknown-block(0,0): error -6
Please append a correct "root=" boot option; here are the available partitions:
1f00          131072 mtdblock0  (driver?)
1f01           32768 mtdblock1  (driver?)
Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
CPU: 0 PID: 1 Comm: swapper/0 Not tainted 4.8.0-rt1+ #1
Hardware name: ARM-Versatile Express
[<8010eb98>] (unwind_backtrace) from [<8010b648>] (show_stack+0x10/0x14)
[<8010b648>] (show_stack) from [<80371b88>] (dump_stack+0x80/0xa8)
[<80371b88>] (dump_stack) from [<801aeed0>] (panic+0xf4/0x2c8)
[<801aeed0>] (panic) from [<80901720>] (mount_block_root+0x1f0/0x310)
[<80901720>] (mount_block_root) from [<80901974>] (mount_root+0x134/0x168)
[<80901974>] (mount_root) from [<80901b10>] (prepare_namespace+0x168/0x1f0)
[<80901b10>] (prepare_namespace) from [<8090125c>] (kernel_init_freeable+0x2c0/0x318)
[<8090125c>] (kernel_init_freeable) from [<805ff348>] (kernel_init+0x8/0x14c)
[<805ff348>] (kernel_init) from [<801078e0>] (ret_from_fork+0x14/0x34)
---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
```

Figure 37: Échec du lancement du process init suite au lancement du noyau Linux

.7. Création de rootfs (busybox) résidant en RAM

D’une façon générale, le système de fichier qui constitue les applications et les services au démarrage après avoir lancer le noyau, se trouvent dans un espace de stockage classique tels que une partition d’un disque dur ou d’un carte micro SD. Le premier processus (parent de tout les processus est soit init (sysVinit) ou systemd.

Il existe de nombreuses raisons pour créer un système de fichiers basé sur la mémoire sous Linux, dont la moindre n'est pas de fournir une latence presque nulle et une zone extrêmement rapide aux fichiers temporaires.

Une utilisation principale d'un disque RAM est pour les répertoires de mise en cache des applications ou les zones de travail.

La création d'un système de fichiers racine qui sera mis en mémoire ram (ramdisk) implique les trois étapes suivantes :

- Une ré-configuration du noyau linux pour qu’il prends en charge ce type de système de fichier particulier.
- La cross-compilation de busybox comme étant un logiciel minimal qui combine à peu prés deux cents commandes unix dans un seul binaire accessible via un invité de commande shell
- Le re-lancement de u-boot et permettre le chargement du rootfs via le noyau Linux

.7.a) Reconfiguration du noyau Linux

Afin de permettre au noyau le chargement d’un **rootfs** résidant en mémoire vive RAM, il faut avoir à le reconfigurer convenablement.

A partir de l’emplacement du noyau linux, tapez la commande suivante et commencer la reconfiguration selon les fugures qui vont suivre :

```
make ARCH=arm CROSS_COMPILE= arm-linux-gnueabihf- menuconfig
```

Dans la section générale, activer le support de Initial RAM filesystem et RAM disk

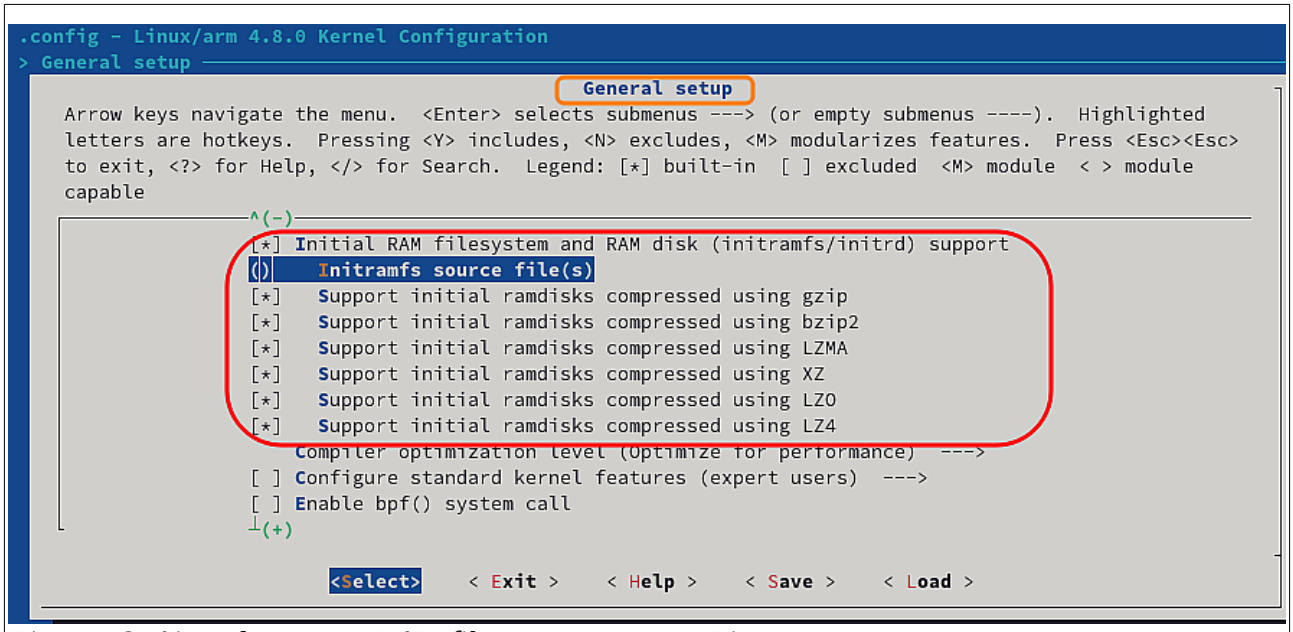

Figure 38: Ajout du support RAM filesystem au noyau Linux

Dans la section globale **Device Drivers** :

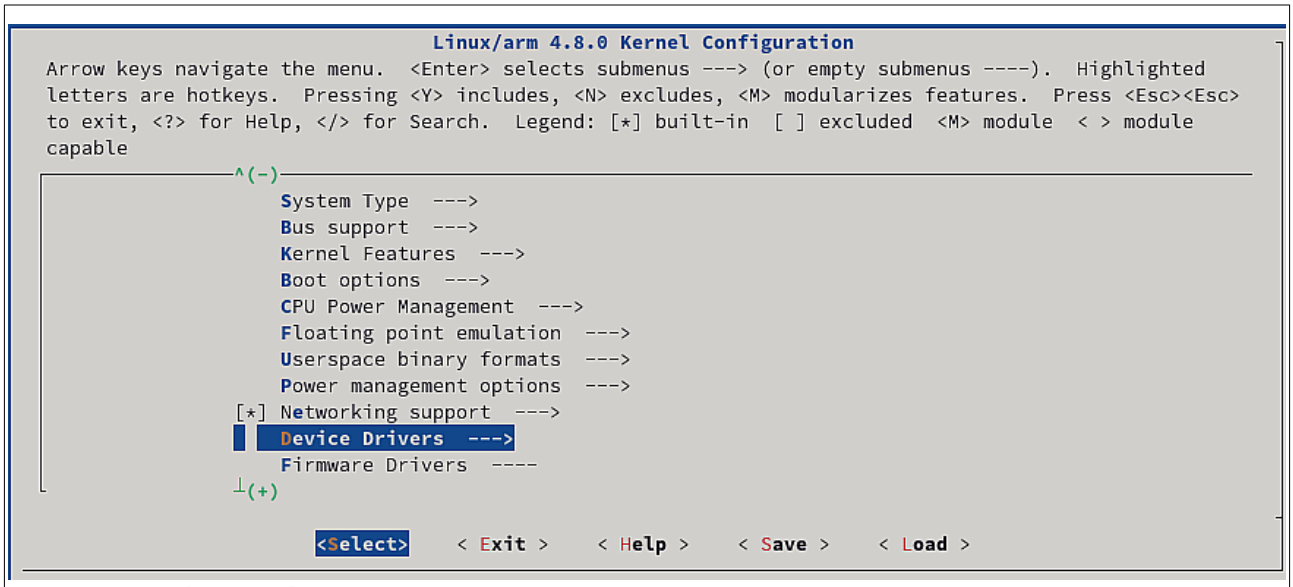

Figure 39: Selection du sous menu Device Drivers

Cocher la case **Block devices**

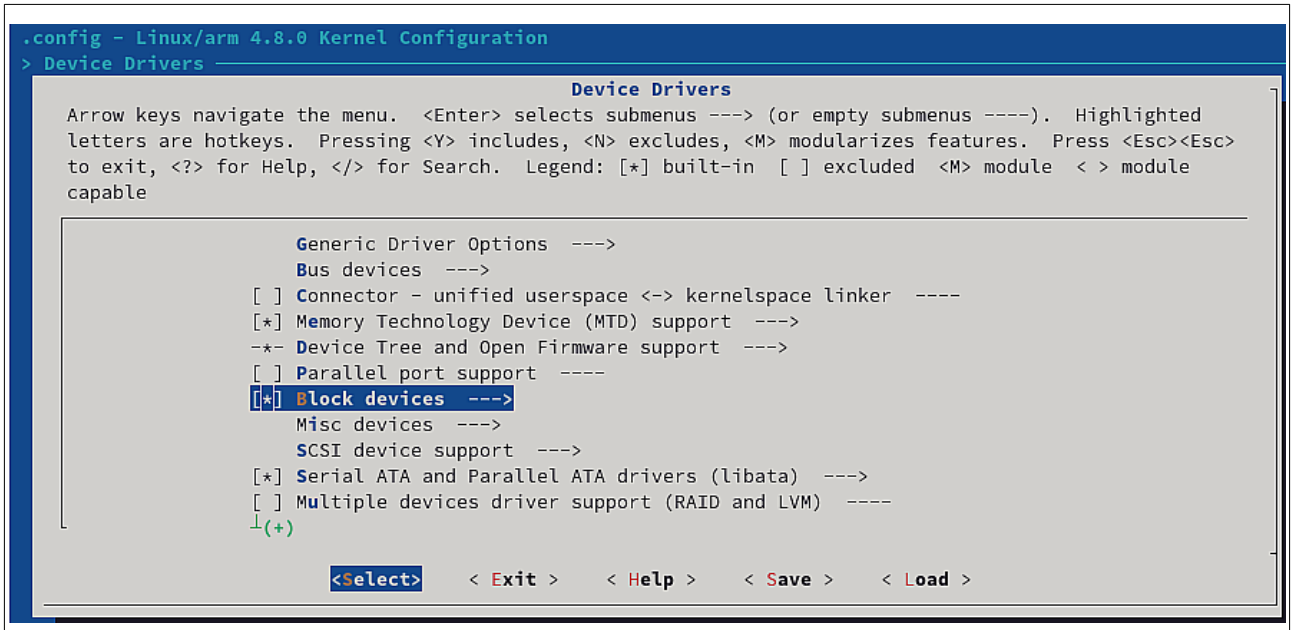

Figure 40: Activation des périphériques communiquant par unité de Bloc

Cocher la case **RAM Block device support** et préciser un nombre de 8 blocs mémoires RAM de taille 8192 Ko chacun

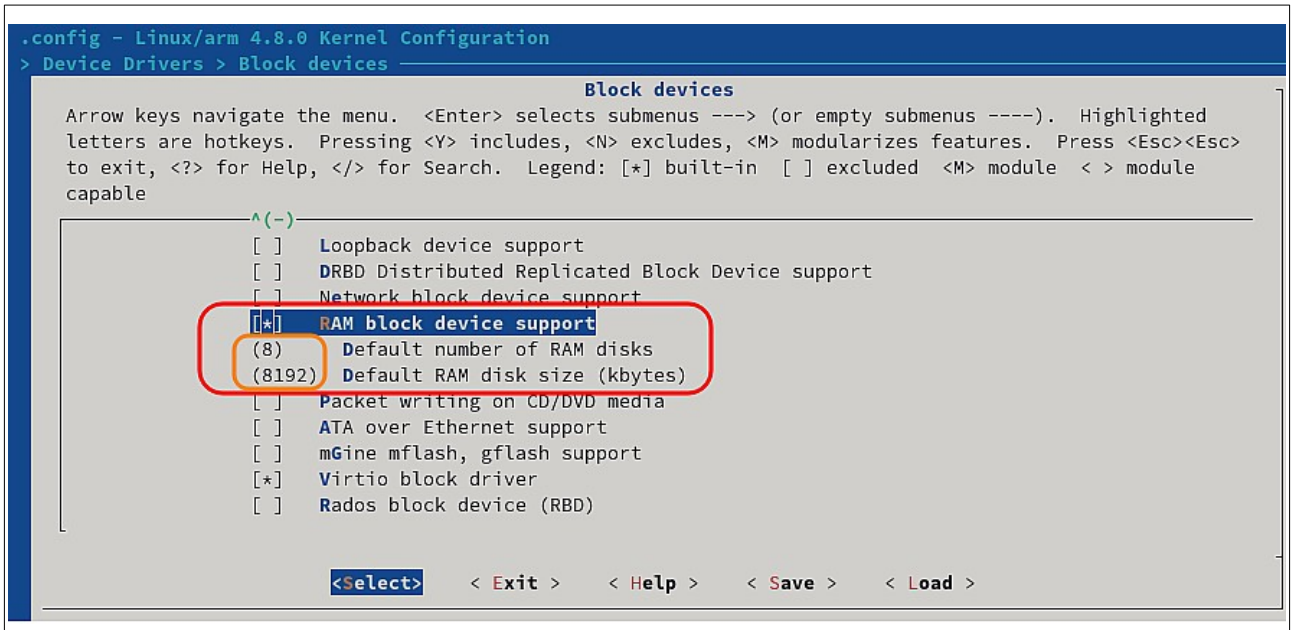

Figure 41: Ajout du support RAM Block Device en précisant le nombre de blocs en mémoire vive

Ajouter la prise en charge du système de fichiers EXT2. ramdisk est une technologie de disque virtuel pour la mémoire, qui n'est pas essentiellement un système de fichiers. Ramdisk utilise le format ext2 en mémoire

Pour ceci, sélectionner dans le menu générale du noyau la section **File Systems**

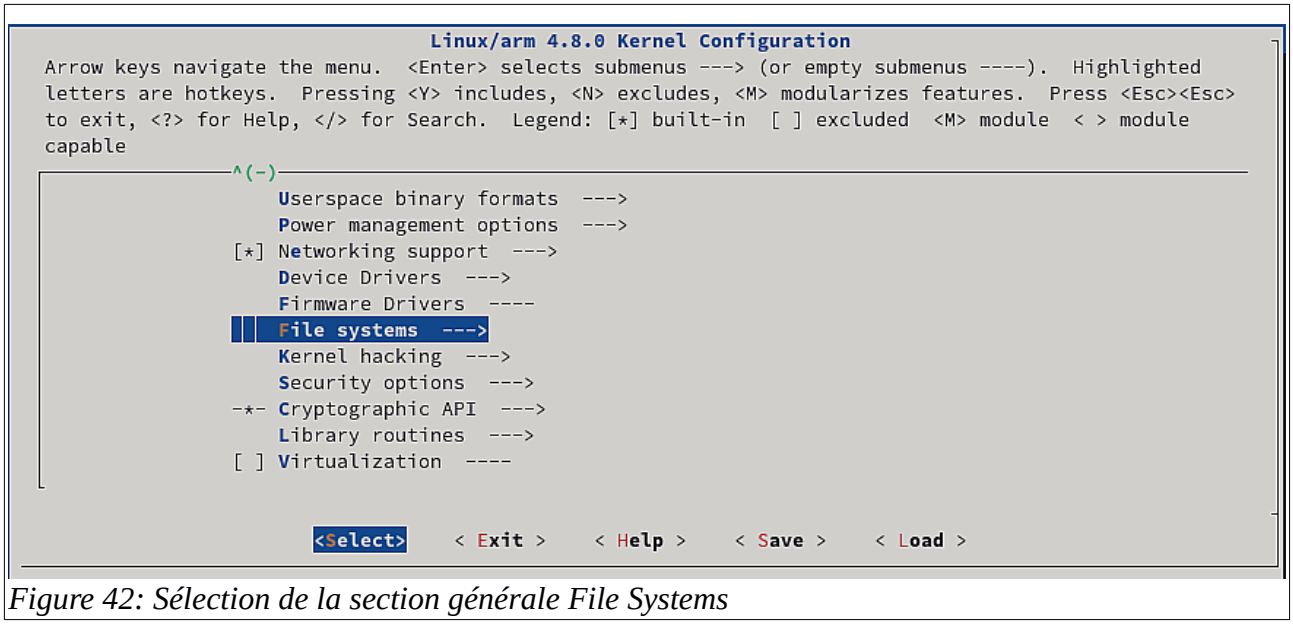

Figure 42: Sélection de la section générale File Systems

Dedans laquelle, activer le support du fs ext2 et d'autres détails :

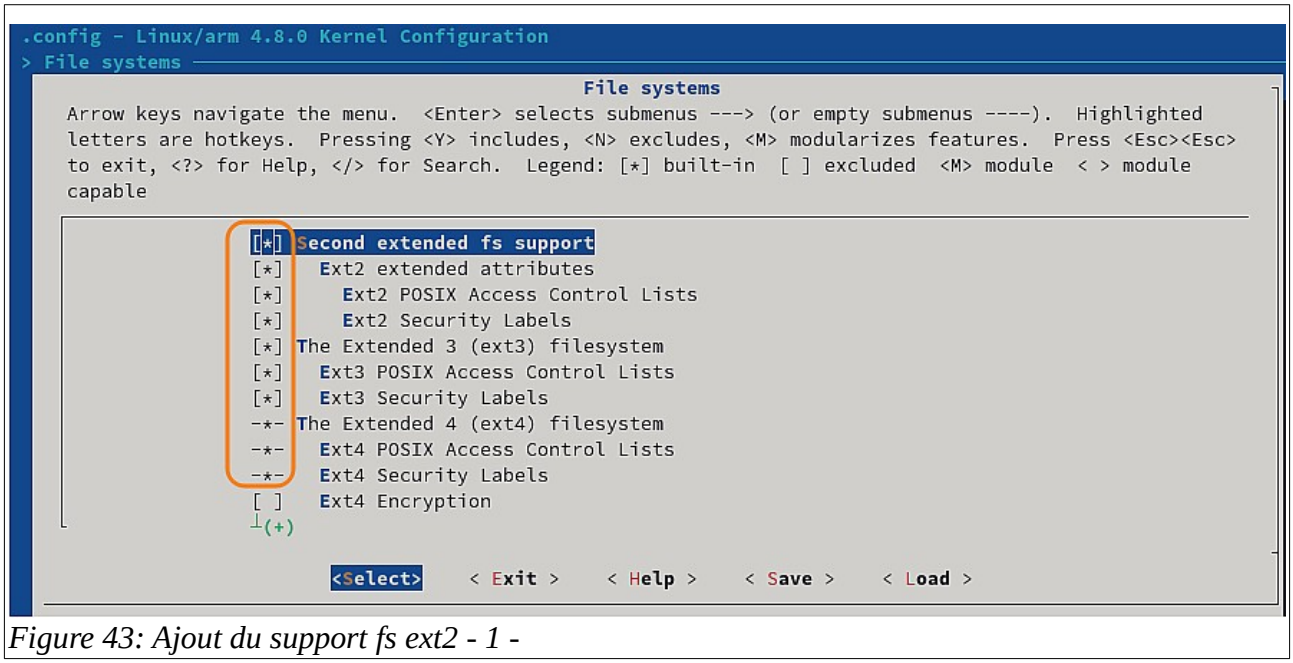

Figure 43: Ajout du support fs ext2 - 1 -

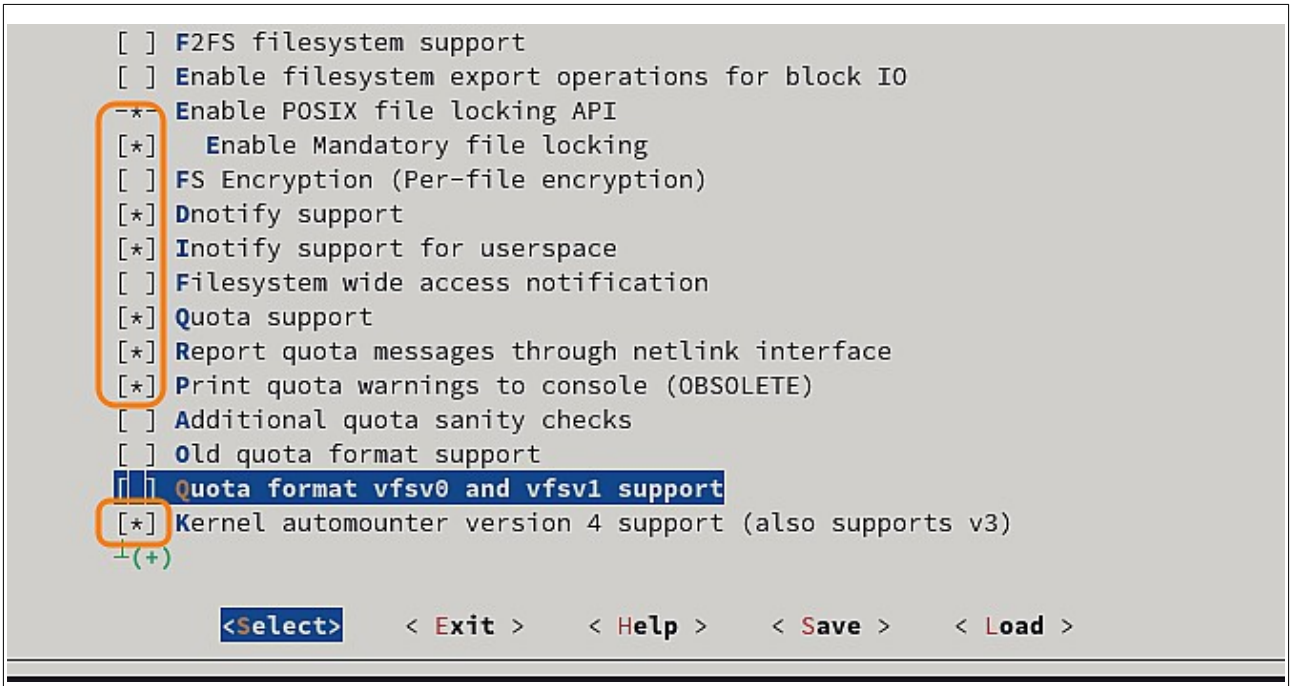

Figure 44: Ajout du support fs ext2 - 2 -

Après avoir sauvegarder cette configuration du noyau linux, il suffit de recross-compiler avec la commande :

```
make ARCH=arm CROSS_COMPILE= arm-linux-gnueabihf- all -j`nproc`
```

.7.b) Cross-compilation de BusyBox

Après avoir configurer le noyau Linux afin de permettre le support de RAMFS et de EXT2, maintenant l'image qui va résider en RAM et charger au démarrage via le noyau devra contenir la suite des choses.

L'image **ramdisk.gz** pourra contenir un simple binaire exécutable qui affiche un message de bonjour, comme ça pourra contenir tout un rootfs, ça veut dire des répertoires systèmes : / , etc, bin, usr, home, root, sys,dev, tmp etc …

Le noyau linux est paramétré afin de lancer un programme nommé init ou linuxrc avec un paramètre nommé init=……

Afin de combler la suite du système d'exploitation (le processus init, les services au démarrage et au moins un invité de commande), on pourra opter à la solution open source nommé **BusyBox** qui fournit un environnement assez complet pour tout petit système ou système embarqué.

Via un terminal, télécharger busybox, décompresser les sources et procéder à la configuration :

```
git clone git://busybox.net/busybox.git busybox
git branch -a # la branche stable et la plus récente est : remotes/origin/1_34_stable
git checkout -b remotes/origin/1_34_stable
cd busybox
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

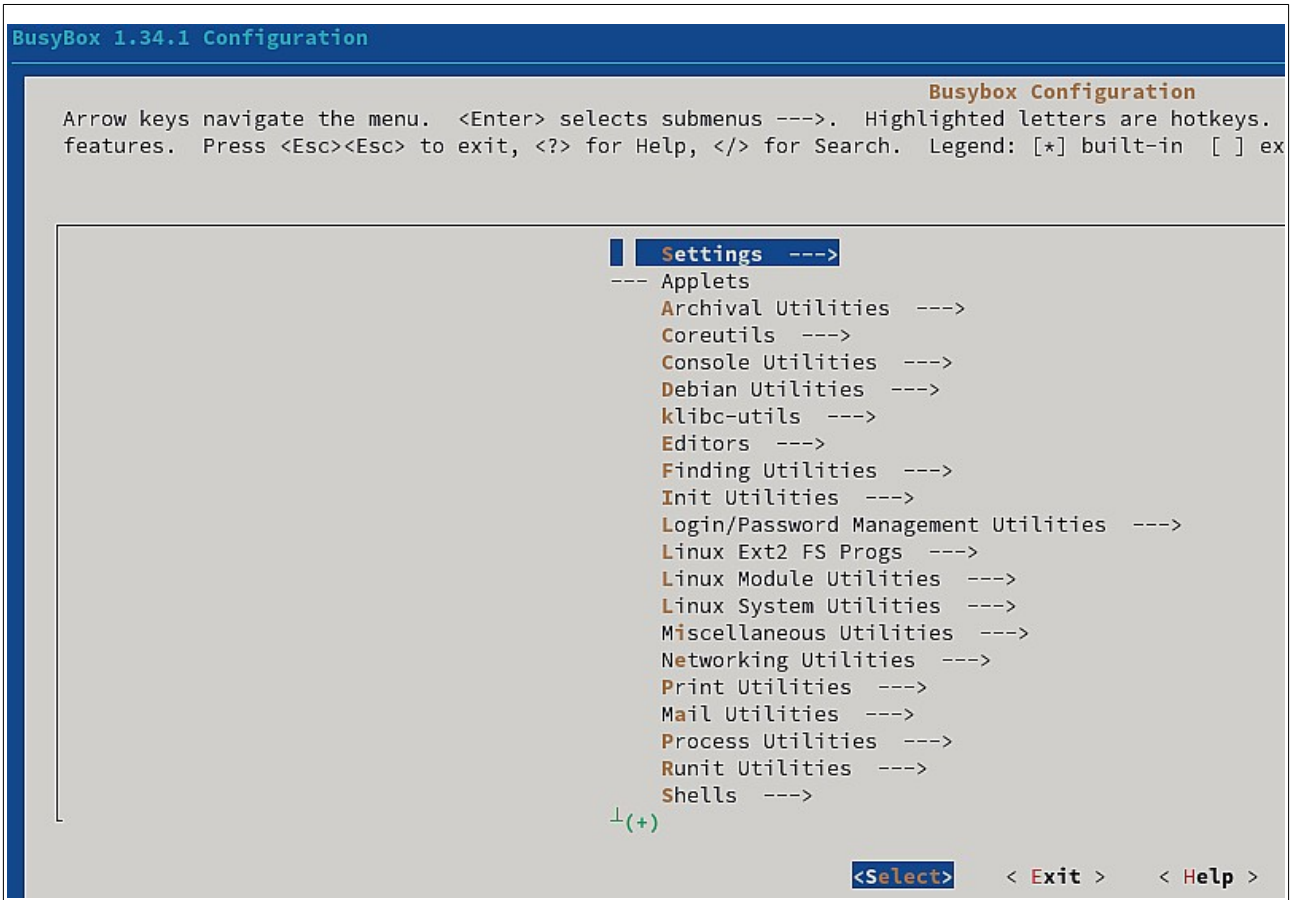

Figure 45: Menu de configuration globale de BusyBox

A partir du menu globale de busybox, vous pouvez choisir différents utilitaires et programmes qu'on désire utiliser, une fois le système démarre avec un invité de commande.

Afin de rendre plus facile l'élaboration du système d'exploitation généré, dans la première section ***settings***, activer le lien statique avec les bibliothèques et définir le préfixe du cross-compilateur (arm-linux-gnueabihf-) [facultative]

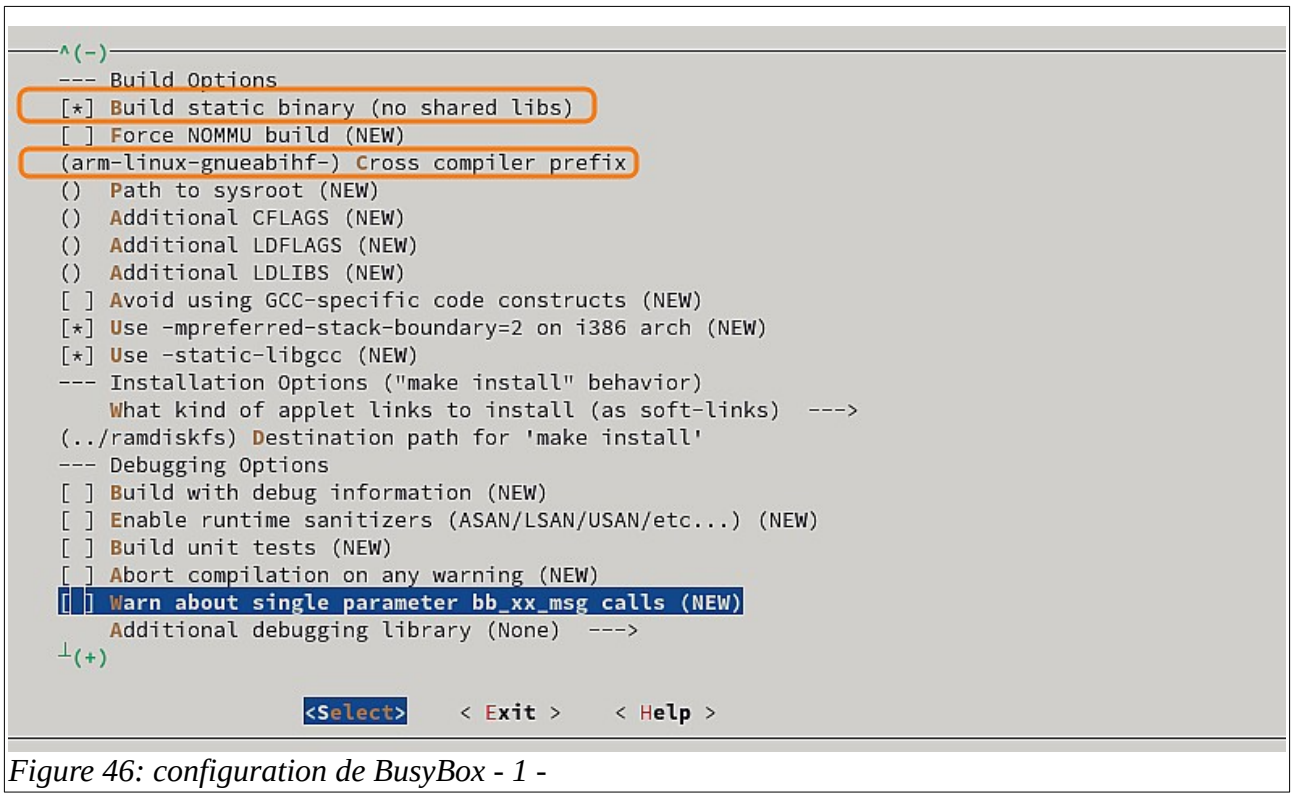

Figure 46: configuration de BusyBox - 1 -

La liaison du fichier bin/busybox avec les bibliothèques est dynamique par défaut. c'est ainsi que le binaire busybox final est beaucoup plus petit car une grande partie du code réside dans un autre fichier : la bibliothèque standard C utilisée lors de la compilation du logiciel qui doit également être présente sur le système de fichiers racine final.

Lorsque vous utilisez une bibliothèque partagée, le système d'exploitation effectue le dernier lien au moment de l'exécution en chargeant à la fois l'exécutable et la bibliothèque partagée et en corrigeant les références dans le programme pour exécuter le code dans la bibliothèque partagée.

Lorsqu'il est lié de manière statique, tout le code utilisé par le programme réside dans le fichier. Le code est chargé en mémoire et exécuté. **C'est une opération beaucoup plus rapide** que le chargement d'un fichier avec des bibliothèques partagées, **en échange d'une plus grande empreinte mémoire**, car une nouvelle copie du code est chargée pour chaque instance de BusyBox exécutée.

Si l'exécutable BusyBox ne contient que quelques exécutables, la règle de base est d'environ une taille de programme de 200 Ko. La liaison statique est alors logique en raison des performances accrues et des économies réalisées en n'ayant pas besoin des bibliothèques partagées.

D'un autre côté, si les autres programmes du système nécessitent des bibliothèques partagées, les utiliser également pour BusyBox est parfaitement logique.

spécifier ensuite l'emplacement dedans lequel BusyBox sera installé : ../ramdiskfs

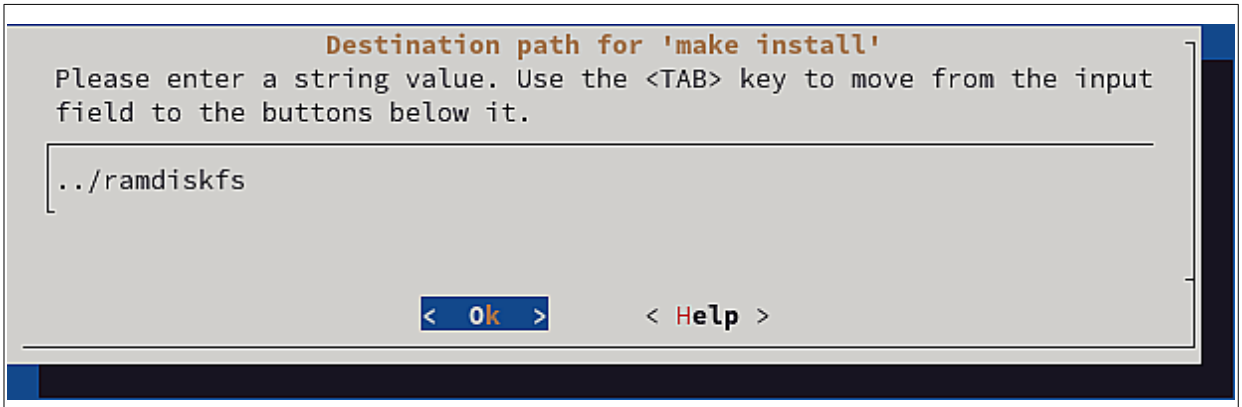

Figure 47: Configuration de BusyBox - 2 -

Une fois, vous quittez le menu de configuration et vous sauvegardez, le fichier de configuration résultant est **.config** .

Lancer la cross-compilation de BusyBox, ensuite l'installation dans l'emplacement ../ramdiskfs

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j`nproc`
make install
```

Comme résultat,la création de quelques répertoires contenant les utilitaires BusyBox :

```
../ramdiskfs//usr/sbin/ubiattach -> ../../bin/busybox
../ramdiskfs//usr/sbin/ubidetach -> ../../bin/busybox
../ramdiskfs//usr/sbin/ubimkvol -> ../../bin/busybox
../ramdiskfs//usr/sbin/ubirename -> ../../bin/busybox
../ramdiskfs//usr/sbin/ubirmvol -> ../../bin/busybox
../ramdiskfs//usr/sbin/ubirsvol -> ../../bin/busybox
../ramdiskfs//usr/sbin/ubiupdatevol -> ../../bin/busybox
../ramdiskfs//usr/sbin/udhcpd -> ../../bin/busybox


-----------------------------------------------------
You will probably need to make your busybox binary
setuid root to ensure all configured applets will
work properly.
-----------------------------------------------------

anouarchemek@f060e729ee71:~/fsl-community-bsp/TP2/busybox-1.34.1$ cd ../ramdiskfs/
anouarchemek@f060e729ee71:~/fsl-community-bsp/TP2/ramdiskfs$ ls -l
total 4
drwxr-xr-x. 1 anouarchemek anouarchemek 966 Nov 20 20:45 bin
lrwxrwxrwx. 1 anouarchemek anouarchemek  11 Nov 20 20:45 linuxrc -> bin/busybox
drwxr-xr-x. 1 anouarchemek anouarchemek 986 Nov 20 20:45 sbin
drwxr-xr-x. 1 anouarchemek anouarchemek  14 Nov 20 20:45 usr
anouarchemek@f060e729ee71:~/fsl-community-bsp/TP2/ramdiskfs$ file bin/busybox
bin/busybox: ELF 32-bit LSB executable, ARM, EABI5 version 1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0,
, stripped
anouarchemek@f060e729ee71:~/fsl-community-bsp/TP2/ramdiskfs$
```

Figure 48: Résultat d'installation de BusyBox après cross-compilation

Comme, indiqué dans la figure ci-dessus, les dossiers qui compose tout le FHS (FileSystem Hierarchy Standard) sont manquants, on doit ajouter quelques fichiers et répertoires systèmes nécessaires au bon démarrage.

Créer les répertoires suivants :

```
cd ../ramdiskfs/
sudo mkdir mnt tmp var sys proc etc lib dev root home
sudo mkdir etc/init.d/
```

Figure 49: Ajouts de dossiers nécessaires pour rootfs

Comme étant l'utilisateur root, créer et peupler le fichier **etc/fstab**

```
#device      mount-point    type    options      dump   fsck order
proc             /proc            proc     defaults                0       0
tmpfs            /tmp             tmpfs    defaults                0       0
sysfs            /sys             sysfs    defaults                0       0
tmpfs            /dev             tmpfs    defaults                0       0
var              /dev             tmpfs    defaults                0       0
ramfs            /dev             ramfs    defaults                0       0
debugfs       /sys/kernel/debug   debugfs  defaults                0       0
```

Comme étant l'utilisateur root, créer et peupler le script **etc/init.d/rcS**

```
#!/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel
mount -a
echo /sbin/mdev /proc/sys/kernel/hotplug
mdev -s
```

N'oublier pas d'ajouter le droit d'exécution par rapport au fichier **etc/init.d/rcS**

```
sudo chmod +x etc/init.d/rcS
```

Comme étant l'utilisateur root, créer et peupler le fichier **etc/inittab** qui est le fichier de configuration du processus init

```
# /etc/inittab
::sysinit:/etc/init.d/rcS
ttyAMA0::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
::restart:/sbin/init
```

Comme étant l'utilisateur root, créer et peupler le script **etc/profile**

```
# /etc/profile: system-wide .profile file for the Bourne shells
PATH="/usr/bin:/usr/sbin:/bin:/sbin"
export PATH
umask 027
echo "Welcome to BusyBox GNU/Linux-$(uname -r)"
echo 2 >/proc/sys/kernel/randomize_va_space
echo 8192 >/proc/sys/vm/min_free_kbytes
```

Tout les fichiers et répertoires que vous avez ajoutés ou qui viennent avec BusyBox, devrons avoir comme propriétaire **root** et comme groupe propriétaire **root** aussi

```
cd  ramdiskfs

sudo chown -R root *

sudo chown -R :root *
```

Enfin, il suffit de créer une image ramfs à partir de ce que a été construit et formaté avec le système de fichier ext2 (pas besoin de journalisation, le système d'exploitation résidera en mémoire RAM)

```
sudo dd if=/dev/zero of=ramdisk bs=1024 count=8192

sudo mke2fs -F -m0 ramdisk #Format, -F force, -m0 do not reserve space for administrator

sudo mount -t ext2 ramdisk /mnt/loop/

sudo cp  -afR ramdiskfs/* /mnt/loop/

sudo umount /mnt/loop/

sudo gzip -v9 ramdisk  #Generate ramdisk.gz
```

```
~/w/TP2  sudo dd if=/dev/zero of=ramdisk bs=1024 count=8192
8192+0 enregistrements lus
8192+0 enregistrements écrits
8388608 octets (8,4 MB, 8,0 MiB) copiés, 0,140872 s, 59,5 MB/s
~/w/TP2  sudo mke2fs -F -m0 ramdisk
mke2fs 1.46.3 (27-Jul-2021)
Rejet des blocs de périphérique : complété
En train de créer un système de fichiers avec 8192 1k blocs et 2048 i-noeuds.

Allocation des tables de groupe : complété
Écriture des tables d'i-noeuds : complété
Écriture des superblocs et de l'information de comptabilité du système de
fichiers : complété

~/w/TP2  sudo mount -t ext2 ramdisk /mnt/loop/
~/w/TP2  sudo rm -fr /mnt/loop/lost+found/
~/w/TP2  sudo cp  -afR ramdiskfs/* /mnt/loop/
~/w/TP2  tree -L 1 /mnt/loop/
/mnt/loop/
├── bin
├── dev
├── etc
├── home
├── lib
├── linuxrc -> bin/busybox
├── mnt
├── proc
├── root
├── sbin
├── sys
├── tmp
├── usr
└── var

13 directories, 1 file
~/w/TP2  sudo umount /mnt/loop/
~/w/TP2  gzip -v9 ramdisk
gzip: ramdisk.gz: Permission denied
 !  ~/w/TP2  sudo gzip -v9 ramdisk
ramdisk:         86.7% -- replaced with ramdisk.gz
~/w/TP2  ls -lh ramdisk.gz
-rw-r--r--. 1 root root 1,1M 21 nov.  01:15 ramdisk.gz
```

Figure 50: Création d'une image ramfs compressé

.7.c) Lancement de u-boot avec prise en compte de ramdiskfs

Dans la section 6, nous avons lancé u-boot via qemu en prenant en considération une interface réseau virtuelle que nous avons nommé tap1. Nous allons relancer l'émulateur avec les mêmes paramètres, mais cette fois ci, on va charger en ram en plus du noyau Linux compressé **zImage** et du fichier **vexpress-v2p-ca9.dtb** le fichier ramfs compressé nommé **ramdisk.gz**

```
anouarchemek@ed579c03f746:/tftpboot$ tree
.
├── ramdisk.gz
├── vexpress-v2p-ca9.dtb
└── zImage

0 directories, 3 files
anouarchemek@ed579c03f746:/tftpboot$
```

Figure 51: Les fichiers d'échange entre la machine host et la machine cible (qemu) via tftp

Relancer l'émulateur qemu-system-arm avec la commande suivante :

```
sudo qemu-system-arm -M vexpress-a9 -m 512M -kernel u-boot -net nic -net tap,ifname=tap1,script=no,downscript=no  -nographic
```

Une fois u-boot lancé via l'émulateur de la carte,rétablissez la configuration réseau avec la machine hôte et configurer le noyau afin qu'il ait les bon argument via la variable **bootargs**

- setenv ipaddr 192.168.1.2
- setenv serverip 192.168.1.1
- setenv netmask 255.255.255.0
- **setenv bootargs 'console=ttyAMA0,38400 root=/dev/ram0 rw init=/linuxrc initrd=0x70000000,0x800000'**
- tftp 0x60000000 vexpress-v2p-ca9.dtb
- tftp 0x60100000 zImage
- tftp 0x70000000 ramdisk.gz
- bootz 0x60100000 - 0x60000000 - 0x70000000

```
Environment size: 4469/262140 bytes
=> setenv ipaddr 192.168.1.2
=> setenv serverip 192.168.1.1
=> setenv netmask 255.255.255.0
=> setevn bootargs 'console=ttyAMA0,115200 root=/dev/ram0 rw init=/linuxrc initrd=0x70000000,0x800000'
Unknown command 'setevn' - try 'help'
=> setenv bootargs 'console=ttyAMA0,115200 root=/dev/ram0 rw init=/linuxrc initrd=0x70000000,0x800000'
```

Figure 52: modification de bootargs afin de permettre le lancement du processus linuxrc depuis la ram

N'oublions pas aussi de transférer le noyau linux **zImage**, le fichier **vexpress-v2p-ca9.dtb** et le **ramdisk.gz** depuis la machine hôte vers la machine émulé via tftp

```
=> tftp 0x70000000 ramdisk.gz
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC 52:54:00:12:34:56
Using ethernet@3,02000000 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.2
Filename 'ramdisk.gz'.
Load address: 0x70000000
Loading: #################################################################
         #################################################################
         #################################################################
         ########################
         1.2 MiB/s
done
Bytes transferred = 1118044 (110f5c hex)
smc911x: MAC 52:54:00:12:34:56
=>
```

Figure 53: Transfert de ramdisk.gz vers la mémoire ram à l'adresse 0x70000000

Depuis le fichier **u-boot.cfg,** le noyau sera chargé à l'adresse 0x60100000 et le fichier vexpress-v2p-ca9.dtb sera chargé à l'adresse 0x60000000 , vous pouvez modifier ceci

(voir section 6) ou garder les mêmes valeurs par défaut. Essentiellement, tachez de charger le **rootfs** dans un emplacement dégagé de la mémoire RAM.

Une fois, le noyau, le fichier .dtb et le rootfs sont chargés via tftp, l'amorçage du système d'exploitation est effectué avec la commande bootz, tout en spécifiant les adresses des différents composants (kernel, dtb et rootfs)

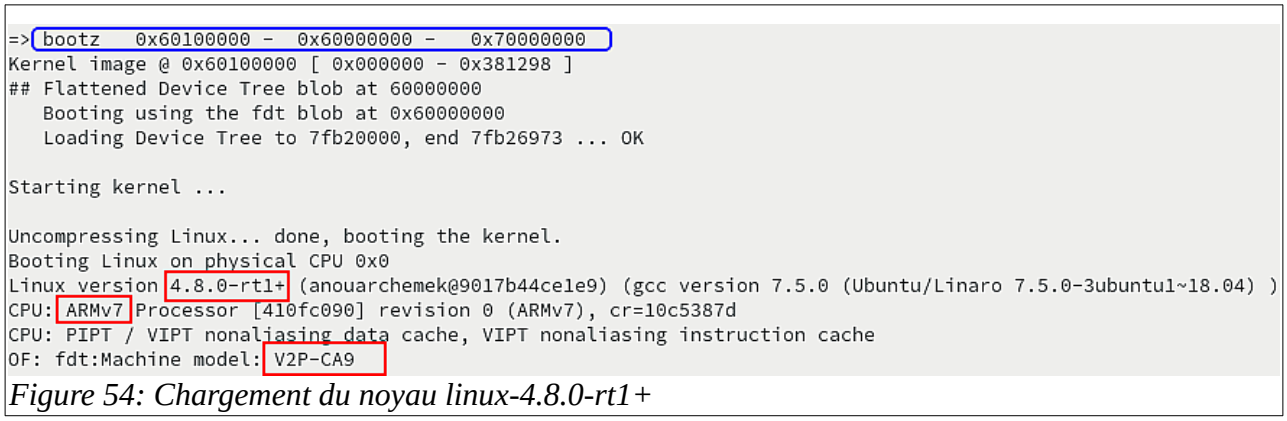

Figure 54: Chargement du noyau linux-4.8.0-rt1+

Aprés plusieurs routines, le noyau linux, tachera de charger le ramfs dépendamment de bootargs et enfin le lancement de bin/Busybox

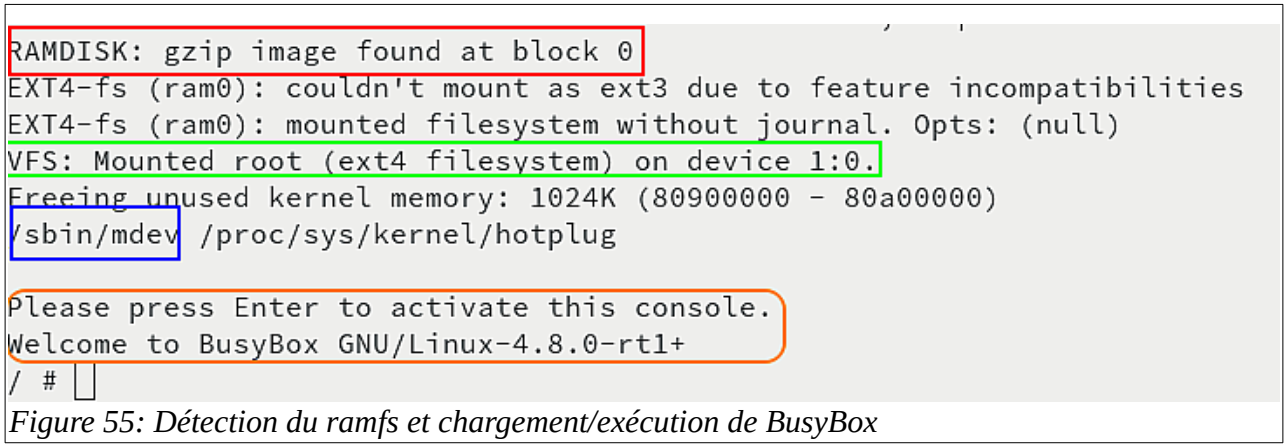

Figure 55: Détection du ramfs et chargement/exécution de BusyBox

On peut tester l'invité de commande avec plusieurs commandes et en activant aussi l'interface réseau eth0 et en essayant la connectivité réseau avec la machine Hôte (Ubuntu 18.04)

```
/ # uname -a
Linux (none) 4.8.0-rt1+ #4 SMP PREEMPT RT Sun Nov 14 10:16:42 UTC 2021 armv7l GNU/Linux
/ # ifconfig eth0 192.168.1.2 netmask 255.255.255.0 up
/ # ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=8.131 ms
^C
--- 192.168.1.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 8.131/8.131/8.131 ms
/ # cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)
BogoMIPS        : 177.92
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc09
CPU revision    : 0

Hardware        : ARM-Versatile Express
Revision        : 0000
Serial          : 0000000000000000
/ #
```

Figure 56: Lancement de commandes via l'invité shell de busybox – 1 –

```
/ # which busybox
/bin/busybox
/ # cd /bin
/bin # ls -l
total 1610
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 arch -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 ash -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 base32 -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 base64 -> busybox
-rwxr-xr-x    1 1000     1000       1546132 Nov 20 20:45 busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 cat -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 chattr -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 chgrp -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 chmod -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 chown -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 conspy -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 cp -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 cpio -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 cttyhack -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 date -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 dd -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 df -> busybox
lrwxrwxrwx    1 1000     1000             7 Nov 20 20:45 dmesg -> busybox
```

Figure 57: Lancement de commandes via l'invité shell de busybox – 2 –