# Labo 1

## Ressources used

`Lab/Lab1-System/CustomSystem`
⇒ Go to `CustomSystem`
⇒Decompress files with command:

```shell
for file in *.tar*; do
    tar -xf "$file"
done
```

## Contenu Lab

### Step 1: Manually compiling the image

In the first labo there is compilation and creation of the elements required by the boot loader and the OS. In the end we get all the files that are used to load the OS on the beagle bone.

- **Compilation U-Boot**
- **Compilation Kernel**
- **Compilation BusyBox**
- **Creating Root Filesystem**
- **Creating the final image from the previous elements**

The following commands are used for that:

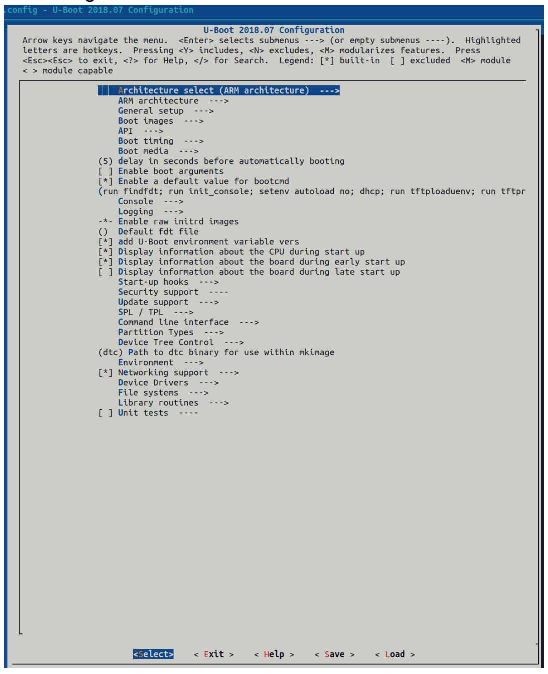
```shell
cd u-boot-2018.07/
patch -p1 < ../ele674_uboot_patches/ele674_20200901.patch
cp ../.config_uboot_ele674 .config
make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- all

cd ../KERNEL/
cp ../.config_ti-linux-rt-4.14.y .config
make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- zImage
make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- modules
make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- dtbs

cd ../busybox-1.31.1/
cp ../.config_busybox_ele674 .config
make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi-

cd ..
mkdir RootFS
cd RootFS
mkdir dev proc sys etc lib bin sbin usr root mnt tmp var
mkdir etc/init.d usr/bin usr/sbin usr/lib var/log

cd ../KERNEL/
make -j12 ARCH=arm INSTALL_MOD_PATH=../RootFS modules_install

cd ../busybox-1.31.1/
make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- CONFIG_PREFIX=../RootFS install

cd ../RootFS/
cp -r ../temp/lib/* lib/.
cp -r ../temp/usr/lib/* usr/lib/.
cp -r ../temp/etc/* etc/.
rm lib/modules/4.14.108/build
rm lib/modules/4.14.108/source

cd ..
sudo makeimage.sh RootFS
```

#### Commands explanation

1. **U-Boot Build:**
   - `cd u-boot-2018.07/`: Change to the U-Boot source directory.
   - `patch -p1 < ../ele674_uboot_patches/ele674_20200901.patch`: Apply a patch to U-Boot source code.
   - `cp ../.config_uboot_ele674 .config`: Copy a pre-existing configuration file for U-Boot.

- `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- all`: Build U-Boot with 12 parallel jobs for the ARM architecture using a specified cross-compiler.

  ⓘ Info

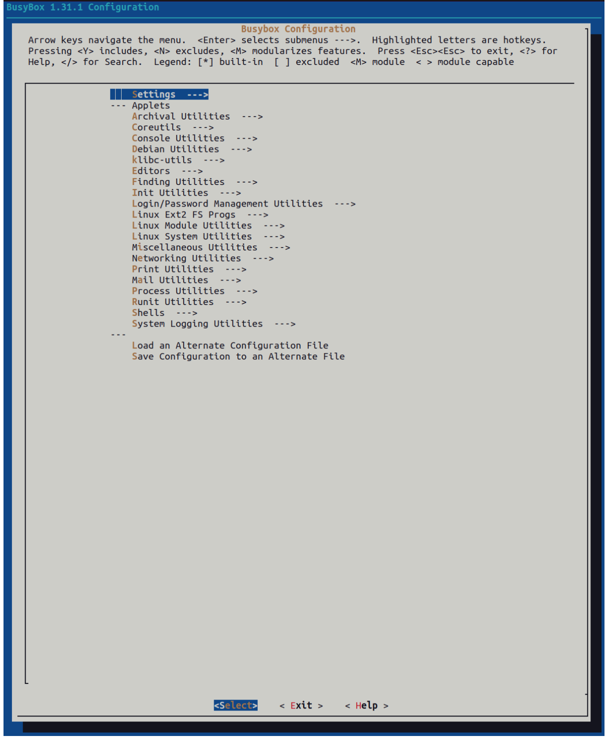
  By default the `all` option will compile and load configuration from the `.config` file.
  ⇒ It is also possible to open a ***GUI*** to add/remove some of these options with the command `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- menuconfig`
  We then get the GUI:
  

```
.config - U-Boot 2018.07 Configuration

                          U-Boot 2018.07 Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press
  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module
  < > module capable

                Architecture select (ARM architecture)  --->
                ARM architecture  --->
                General setup  --->
                Boot images  --->
                API  --->
                Boot timing  --->
                Boot media  --->
            (5) delay in seconds before automatically booting
            [ ] Enable boot arguments
            [*] Enable a default value for bootcmd
            (run findfdt; run init_console; setenv autoload no; dhcp; run tftploaduenv; run tftpr
                Console  --->
                Logging  --->
            -*- Enable raw initrd images
            ()  Default fdt file
            [*] add U-Boot environment variable vers
            [*] Display information about the CPU during start up
            [*] Display information about the board during early start up
            [ ] Display information about the board during late start up
                Start-up hooks  --->
                Security support  ----
                Update support  --->
                SPL / TPL  --->
                Command line interface  --->
                Partition Types  --->
                Device Tree Control  --->
            (dtc) Path to dtc binary for use within mkimage
                Environment  --->
            [*] Networking support  --->
                Device Drivers  --->
                File systems  --->
                Library routines  --->
            [ ] Unit tests  ----

                 <Select>    < Exit >    < Help >    < Save >    < Load >
```

2. **Kernel Build:**
   - `cd ../KERNEL/`: Change to the Linux kernel source directory.
   - `cp ../.config_ti-linux-rt-4.14.y .config`: Copy a configuration file for the specific version of the Linux kernel.
   - `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- zImage`: Build the kernel image (zImage). This is a format that can be loaded by bootloaders.
   - `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- modules`: Build kernel modules. Modules are pieces of code that can be loaded into the kernel on demand, allowing for a modular approach to kernel functionality.
   - `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- dtbs`: Build device tree blobs (DTBs) needed for booting. Device Tree is a data structure used to describe the hardware components of a system to the Linux kernel. DTBs are essential for ARM devices to inform the kernel about the hardware configuration.
3. **BusyBox Build:**
   - `cd ../busybox-1.31.1/`: Change to the BusyBox source directory.
   - `cp ../.config_busybox_ele674 .config`: Copy a BusyBox configuration file.
   - `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi-`: Build BusyBox.

ⓘ Info

By default the `all` option compile and load configuration from the `.config` file.
⇒ It is also possible to open a ***GUI*** to add/remove some of these options with the command `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- menuconfi`
We then get the GUI:



```
BusyBox 1.31.1 Configuration
                              Busybox Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.
 Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for
 Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

            Settings  --->
        --- Applets
            Archival Utilities  --->
            Coreutils  --->
            Console Utilities  --->
            Debian Utilities  --->
            klibc-utils  --->
            Editors  --->
            Finding Utilities  --->
            Init Utilities  --->
            Login/Password Management Utilities  --->
            Linux Ext2 FS Progs  --->
            Linux Module Utilities  --->
            Linux System Utilities  --->
            Miscellaneous Utilities  --->
            Networking Utilities  --->
            Print Utilities  --->
            Mail Utilities  --->
            Process Utilities  --->
            Runit Utilities  --->
            Shells  --->
            System Logging Utilities  --->
        ---
            Load an Alternate Configuration File
            Save Configuration to an Alternate File

                  <Select>    < Exit >    < Help >
```

4. **Root Filesystem Setup:**
   - `cd ..`: Go back to the parent directory.
   - `mkdir RootFS`: Create a directory for the root filesystem.
   - `cd RootFS`: Change to the new RootFS directory.
   - Create various directories (`dev`, `proc`, `sys`, etc.) necessary for a Linux filesystem.
   - `cd ../KERNEL/`: Return to the kernel directory.
   - `make -j12 ARCH=arm INSTALL_MOD_PATH=../RootFS modules_install`: Install kernel modules into the RootFS directory.
5. **BusyBox Installation:**
   - `cd ../busybox-1.31.1/`: Return to the BusyBox directory.
   - `make -j12 ARCH=arm CROSS_COMPILE=/usr/src/gcc-linaro-7.5/bin/arm-linux-gnueabi- CONFIG_PREFIX=../RootFS install`: Install BusyBox into the RootFS.
6. **Copy Additional Files:**
   - `cd ../RootFS/`: Change to the RootFS directory.
   - Copy libraries and configuration files from a temporary directory into the appropriate locations in RootFS.
   - Remove build and source links from the kernel modules directory to clean up.
7. **Create Final Image:**
   - `cd ..`: Go back to the parent directory.
   - `sudo makeimage.sh RootFS`: Run a script to create a filesystem image from the RootFS directory.

### Step 2: Compiling the image using Buildroot

The last step was quite laborious, and takes a lot of time.
⇒ That's why tools such as ***BuildRoot*** were done.

Using these tool, it is now possible to manually select the packages/ configurations you want to use for the différents blocks used for booting (U-boot, kernel, busybox, other modules...).

### Ressources used

⇒ Go to `buildroot-202xxx` directory.
There will be a default .config file containing the default configuration.

**Defining the configuration using a GUI**

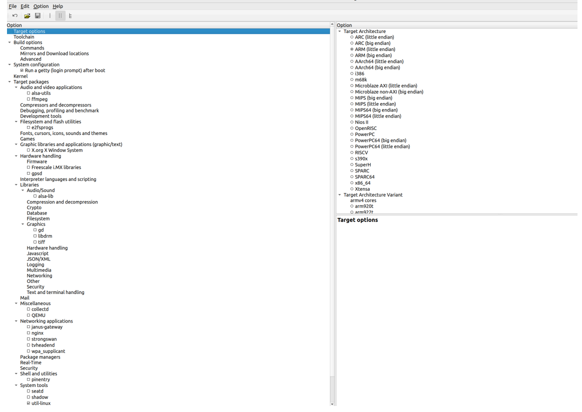
It is also possible to edit with a gui. For that there is two possibility:

```shell
make xconfig
```

The other one is:

```shell
make menuconfig
```

Both are very similar and will open a gui configuration windows in which you can select which modules/options you want to use.



```
Option
  Target options
  Toolchain
▾ Build options
    Commands
    Mirrors and Download locations
    Advanced
▾ System configuration
    ☒ Run a getty (login prompt) after boot
  Kernel
▾ Target packages
  ▾ Audio and video applications
      ☐ alsa-utils
      ☐ ffmpeg
    Compressors and decompressors
    Debugging, profiling and benchmark
    Development tools
  ▾ Filesystem and flash utilities
      ☐ e2fsprogs
    Fonts, cursors, icons, sounds and themes
    Games
  ▾ Graphic libraries and applications (graphic/text)
      ☐ X.org X Window System
  ▾ Hardware handling
      Firmware
      ☐ Freescale i.MX libraries
      ☐ gpsd
    Interpreter languages and scripting
  ▾ Libraries
    ▾ Audio/Sound
        ☐ alsa-lib
      Compression and decompression
      Crypto
      Database
      Filesystem
    ▾ Graphics
        ☐ gd
        ☐ libdrm
        ☐ tiff
      Hardware handling
      Javascript
      JSON/XML
      Logging
      Multimedia
      Networking
      Other
      Security
      Text and terminal handling
    Mail
  ▾ Miscellaneous
      ☐ collectd
      ☐ QEMU
  ▾ Networking applications
      ☐ janus-gateway
      ☐ nginx
      ☐ strongswan
      ☐ tvheadend
      ☐ wpa_supplicant
    Package managers
    Real-Time
    Security
  ▾ Shell and utilities
      ☐ pinentry
  ▾ System tools
      ☐ seatd
      ☐ shadow
      ☒ util-linux
```

```
Option
▾ Target Architecture
    ○ ARC (little endian)
    ○ ARC (big endian)
    ◉ ARM (little endian)
    ○ ARM (big endian)
    ○ AArch64 (little endian)
    ○ AArch64 (big endian)
    ○ i386
    ○ m68k
    ○ Microblaze AXI (little endian)
    ○ Microblaze non-AXI (big endian)
    ○ MIPS (big endian)
    ○ MIPS (little endian)
    ○ MIPS64 (big endian)
    ○ MIPS64 (little endian)
    ○ Nios II
    ○ OpenRISC
    ○ PowerPC
    ○ PowerPC64 (big endian)
    ○ PowerPC64 (little endian)
    ○ RISCV
    ○ s390x
    ○ SuperH
    ○ SPARC
    ○ SPARC64
    ○ x86_64
    ○ Xtensa
▾ Target Architecture Variant
    armv4 cores
    ○ arm920t
    ○ arm922t

Target options
```

## Step 3: Chargement de l'image compilée sur le drone

Dans le résultat des opérations/compilations, récupérer les fichiers suivants:

### Récupération des fichiers à déposer sur le drone

1. **am335x-boneblue.dtb** :
   - **Description** : C'est un fichier de blob d'arbre de périphériques (Device Tree Blob).
   - **Fonction** : Il décrit le matériel de la carte AM335x (par exemple, BeagleBone Blue). Le noyau Linux utilise ce fichier pour comprendre la configuration matérielle et les périphériques disponibles.
2. **rootfs.ext2.gz** :
   - **Description** : C'est une image compressée du système de fichiers racine (Root File System) au format ext2.
   - **Fonction** : Elle contient tous les fichiers et répertoires nécessaires pour le système Linux, y compris les utilitaires, bibliothèques et configurations. Lors du démarrage, cette image est décompressée et montée comme système de fichiers racine.
3. **u-boot.img** :
   - **Description** : C'est l'image principale de U-Boot.
   - **Fonction** : U-Boot est un chargeur de démarrage utilisé pour initialiser le matériel et charger le noyau Linux (et éventuellement le système de fichiers) en mémoire au démarrage.
4. **u-boot-spl.bin** :
   - **Description** : C'est l'image du Secondary Program Loader (SPL) de U-Boot.
   - **Fonction** : Le SPL est une version simplifiée de U-Boot, qui est utilisée pour initialiser des systèmes où la mémoire et les ressources sont limitées. Il est généralement chargé en premier, puis il charge l'image principale de U-Boot.
5. **zImage** :
   - **Description** : C'est l'image compressée du noyau Linux.
   - **Fonction** : Le noyau est le cœur du système d'exploitation. L'image zImage est utilisée pour démarrer le système Linux et gérer les ressources matérielles et les processus.

Sure! Here's the translation:

> **Info**
>
> These files are relatively complex to find; they are also located in the `/Lab/Lab1-System/SystemTest` folder.

---

### Loading Files onto the Drone

⇒ Place the files mentioned earlier in the `/tftpboot` directory.
⇒ Then, when the drone starts up, it will automatically look for its bootloader and image at this location.

### Serial Connection with the Drone

⇒ Open the *Serial Port Terminal* application and select the following settings:

- Port: /dev/ttyUSB0
- Baud rate: 115200
  ⇒ Then, power off and power on the drone: The drone will then fetch the bootloader, the image, and other configurations from the `/tftpboot` folder of the connected machine.
  ⇒ It will then boot up. Messages will normally be displayed in the terminal indicating the different phases of the boot process.

> **Info**
>
> It is possible to interrupt the boot process and modify options at this point.

⇒ When prompted for a login, enter `root`.