TP 3 : Construction d'un système linux embarqué complet

1 Introduction

Les ordinateurs embarqués fonctionnant sous le système d'exploitation Linux sont massivement présents dans les technologies modernes (transports, multimédia, téléphonie mobile, appareils photos …).

Contrairement aux versions de Linux destinées aux ordinateurs personnels et aux serveurs, les différents systèmes Linux embarqués sont conçus pour des systèmes aux ressources limitées.

Les systèmes embarqués sous Linux disposent généralement de peu de RAM et utilisent fréquemment de la mémoire flash plutôt qu'un disque dur. Comme ils sont souvent dédiés à un nombre de tâches réduites sur une cible matérielle bien définie, ils utilisent plutôt des versions du noyau Linux optimisées pour des contextes précis.

Les nécessités de déterminisme et de réactivité entraine parfois les concepteurs à se tourner vers des solutions temps réelles.

L'objectif de ce TP est de réaliser une distribution complète avec mise en oeuvre d'un capteur opérationnel dès sa mise sous tension.

2 Durée

Le TP dure **4h30** mais il est fortement recommandé de prendre connaissance des informations concernant Buildroot et d'effectué les recherches concernant le matériel de la carte Raspberry Pi en amont (parties 4 et 5).

3 Evaluation

L'évaluation portera sur la rédaction d'un compte rendu détaillé :

- explicitant les différentes opérations menées durant le TP.
- présentant une analyse des différents tests mis en œuvre.
  Pensez à prendre des copies d'écran pour illustrer ces opérations.

4 Buildroot

4.1 Introduction

Dans le domaine de l'embarqué, on se retrouve souvent en situation de devoir reconstruire un système complet à partir des sources, pour une architecture cible souvent différente de celle de l'hôte. La cross-compilation et l'organisation d'un système embarqué sont des étapes longues et fastidieuses, surtout lorsque les éléments du système à compiler nécessitent des adaptations. Il existe heureusement des outils libres qui simplifient et accélèrent cette tâche, en proposant généralement des fonctionnalités complémentaires intéressantes.

Buildroot est un de ces outils qui simplifie et automatise le processus de construction d'un système Linux complet pour un système embarqué. Afin d'atteindre cet objectif, Buildroot est capable de générer une chaîne de compilation croisée, un système de fichiers (rootfs), une image du noyau Linux (kernel) et un chargeur de démarrage (firmware/bootloader) pour la cible.

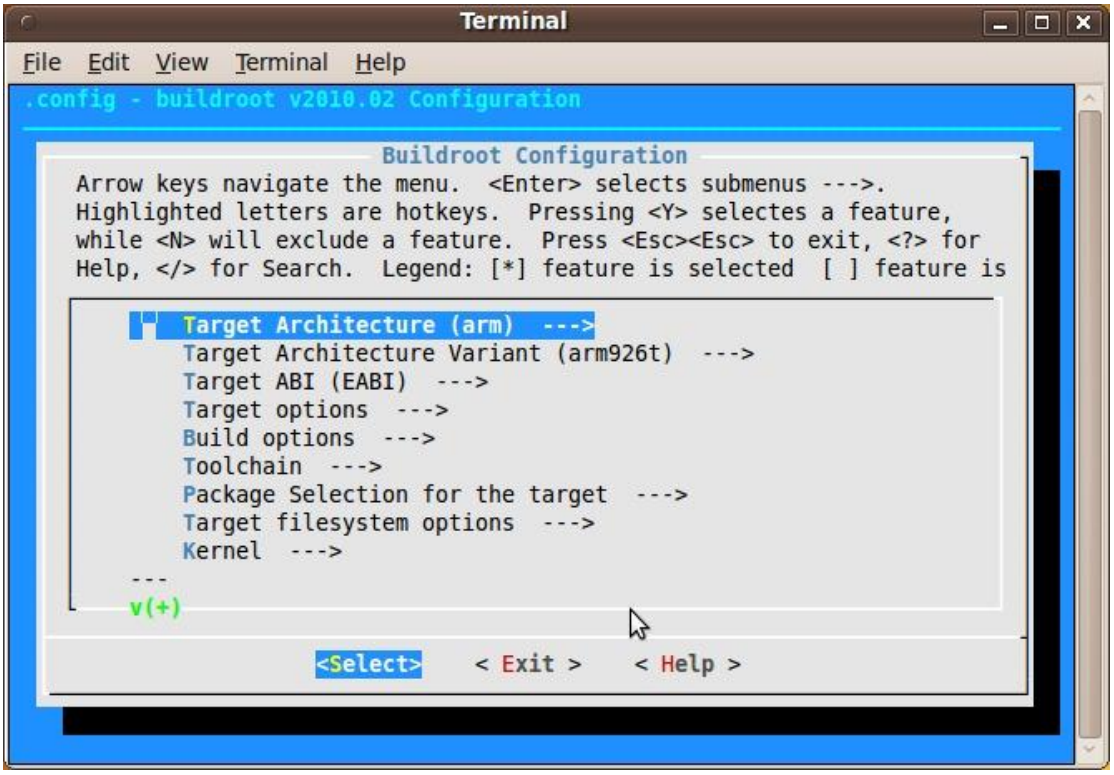
Il prend en charge de nombreux processeurs et leurs variantes (x86, PowerPC, MIPS, ARM, NIOS, etc). Il est également livré avec des configurations par défaut pour un grand nombre de cartes disponibles sur le marché.

4.2 Principe de fonctionnement

Buildroot est techniquement un ensemble de Makefiles définissant, en fonction des options paramétrées par l'utilisateur, la manière de compiler chaque paquet sélectionné avec des options particulières. Il construit finalement une distribution complète et cohérente dont chaque composant a été compilé.

Il possède un outil confortable de configuration, basé et très similaire à celui du noyau Linux : **menuconfig**, que l'on retrouve également avec **Busybox**, **uClibc** et qui peut être utilisé dans tout projet.

```
make menuconfig
```



Une fois que tout est configuré, l'outil de configuration génère un fichier .config qui contient l'ensemble de la configuration. Ce fichier sera lu par le fichier Makefile lors du processus de construction.

Pour démarrer le processus de construction, il suffit de lancer :

```
make
```

La commande **make** effectue généralement les étapes suivantes :

- téléchargement des fichiers sources (si nécessaire);
- configuration, compilation et installation de la chaîne de compilation croisée
- configuration, compilation, corrections (application des patchs) et installation des paquets cibles sélectionnées;
- construction d'une image du noyau
- construction d'une image de bootloader;
- création d'un système de fichiers racine (rootfs) dans les formats sélectionnés.

Le résultat de la construction est stocké dans un répertoire unique, **output/**. Ce répertoire contient plusieurs sous-répertoires :

- **`images/`** : où toutes les images (image du noyau, bootloader et système de fichiers racine) sont stockés. Ce sont les fichiers qui seront copiés sur le système cible.
- **`build/`** : où tous les composants sont construits (ce qui inclut les outils nécessaires par Buildroot sur l'hôte et des paquets compilés pour la cible). Ce répertoire contient un sous-répertoire pour chacun de ces composants.
- **`staging/` :** contient une hiérarchie similaire à une hiérarchie de système de fichiers racine. Ce répertoire contient les en-têtes et les bibliothèques de la chaîne d'outils de compilation croisée et tous les paquets de l'espace utilisateur sélectionnés pour la cible. Cependant, ce répertoire ne vise pas à être le système de fichiers racine pour la cible: il contient un grand nombre de fichiers de développement, les binaires et les bibliothèques qui font qu'il est beaucoup trop grand pour un système embarqué non dénudés. Ces fichiers de développement sont utilisés pour compiler les bibliothèques et les applications pour la cible qui dépendent d'autres bibliothèques.
- **`target/`** : contient presque le système de fichiers **rootfs** complet pour la cible : tout le nécessaire est présent, sauf les fichiers de périphérique dans **/dev** (Buildroot ne peut pas les créer car Buildroot ne fonctionne pas en tant que root). En outre, il ne possède pas les autorisations appropriées (par exemple de **setuid** pour le binaire **busybox**). Par conséquent, ce répertoire ne doit pas être utilisé sur la cible. Par rapport à la mise **`staging/`**, **`target/`** contient uniquement les fichiers et les bibliothèques nécessaires pour exécuter les applications cibles sélectionnés : les fichiers de développement (en-têtes, etc.) ne sont pas présents, les binaires sont dépouillés.
- **`host/`** : contient l'installation des outils compilés pour l'hôte qui sont nécessaires pour la bonne exécution de Buildroot, y compris **la chaîne d'outils de compilation croisée**.

4.3 Cross-compilation toolchain

La toolchain désigne l'ensemble des outils à compiler qui permettra ensuite d'avoir un environnement capable de cross compiler depuis l'architecture host (x86_64) vers l'architecture cible (ARM).

La toolchain regroupe un certain nombre de composants obligatoires comme :

- un compilateur ;
- un linker ;
- un assembleur.

Buildroot propose plusieurs mécanismes pour utiliser une toolchain. Le plus direct est d'utiliser la toolchain interne, et dans ce cas c'est Buildroot qui gérera la création et l'utilisation de la toolchain.

4.4 Init system / bootloader

Les systèmes Linux embarqués sont généralement chargés par le bootloader U-boot, mais ce n'est pas le cas du Raspberry Pi, qui dispose de son propre outil de démarrage qui comprend les fichiers suivants :

- **bootcode.bin** : sert à l'initialisation du GPU
- **bcm2709-rpi-2-b.dtb** : (Device Tree Blob) Fichier de description du matériel. Permet de gérer certaines allocations des ressources et les chargements de modules.
- **start.elf** et **fixup.dat** : bootloader qui termine le processus d'initialisation du CPU et du GPU en attribuant à chacun une partie de la RAM disponible.
- **config.txt** : fichier texte pour personnaliser le comportement du GPU. Entre autres, l'option gpu_mem est très utile puisqu'elle permet de répartir la mémoire disponible entre le GPU et le CPU. Par exemple la ligne « gpu_mem=32 » affecte 32 Mo de Ram au GPU et le reste au CPU.
- **cmdline.txt** : fichier texte qui contient des paramètres supplémentaires à ceux déjà embarqués dans l'image du noyau compilé. Ce fichier est indispensable et ne doit pas être vide. Il contient par défaut les options :
  - **rootwait** : attendre (éventuellement indéfiniment) sans échouer, que la partition contenant l'arborescence des fichiers soit prête ; ceci est nécessaire lorsque l'initialisation du périphérique bloc correspondant peut être longue (notamment pour les disques USB) ;
  - **root=/dev/mmcblk0p2** : la racine de l'arborescence des fichiers se trouve sur la seconde partition de la première (et seule) carte SD ;
- **zImage** : l'image du noyau à démarrer.

4.5 Le noyau

L'élément probablement le plus spécifique d'une plateforme embarquée est le noyau Linux. Contrairement aux kernels fournis avec les distributions pour postes de travail ou serveurs, nous ne voulons pas d'un noyau générique capable de fonctionner sur une multitude de machines différentes, mais d'une configuration bien ajustée, contenant tous les drivers, protocoles, systèmes de fichiers indispensables, sans en ajouter plus que nécessaire.

Le noyau linux standard peut être ajusté en utilisant la commande suivante dans le dossier de Buildroot.

```
make linux-menuconfig
```

Cette opération télécharge, extrait (dans le répertoire output/build/linux-<version>), et pre-configure le kernel avec les options paramétrées dans Buildroot pour la configuration du noyau. La règle **make** ci-dessus rentre dans le répertoire et lance un « **make menuconfig** » avec les options d'environnement pour la cible.

4.6 Le système de fichiers racine rootfs

C'est le système complet installé sur la carte qui sera utilisé après le démarrage du bootloader puis de Linux. Sa configuration s'effectue principalement dans le choix des paquets à installer sur la cible, à partir du menu **Package Selection for the target** de Buildroot.

Il contient également l'ensemble des commandes disponibles sur le futur système. La solution la plus efficace pour un système embarqué est d'utiliser Busybox.

BusyBox est un logiciel qui implémente un grand nombre des commandes standard sous Unix. Il est conçu comme un unique fichier exécutable, ce qui le rend très adapté aux distributions Linux utilisées sur les systèmes embarqués.

Il est notamment très répandu de nos jours sur les périphériques réseaux. On le trouve par exemple sur des points d'accès, des routeurs, des téléphones IP, certains serveurs de stockage en réseau (Network Attached Storage ou NAS) ou encore dans les dernières générations de robots (AR-Drone). En France, le code de BusyBox est également intégré dans les box de certains fournisseurs d'accès Internet : Livebox, Freebox et Neufbox. Il est aussi possible d'installer son propre BusyBox sur Android, très utilisé sur les smartphone et tablettes tactiles, pour obtenir un meilleur contrôle de son matériel.

Pour personnaliser l'ensemble des commandes de Busybox à installer sur la cible :

```
make busybox-menuconfig
```

4.7 La construction

Le processus de construction (**make**) peut être très long. Aussi vaut-il mieux être sur d'avoir bien tout configuré avant de le lancer.

Toutefois, il est possible de relancer une compilation partielle dans certains cas.

Voici quelques règles de base qui peuvent vous aider à comprendre comment travailler avec Buildroot :

- **Lorsque la configuration de l'architecture cible est modifié, une reconstruction complète est nécessaire**. La modification de l'architecture du CPU, le format binaire ou la stratégie de gestion des nombres à virgule flottante par exemple, ont un impact sur l'ensemble du système.
- **Lorsque la configuration de la toolchain est modifiée, une reconstruction complète est généralement nécessaire**. La modification de la configuration de la toolchain implique souvent de modifier la version du compilateur, du type de la bibliothèque C ou sa configuration, ou un autre élément de configuration fondamentale. Ces changements ont un impact sur l'ensemble du système.
- **Quand un paquet supplémentaire est ajouté à la configuration, une reconstruction complète n'est pas forcément nécessaire**. Buildroot détectera que ce paquet n'a jamais été construit, et le construira.
- **Quand un paquet est supprimé de la configuration, Buildroot ne fait rien de spécial**. Il ne supprime pas les fichiers installés par ce paquet du système de fichiers racine de la cible**. Une reconstruction complète est nécessaire pour se débarrasser de ce paquet.**
- **Lorsque les sous-options d'un paquet sont modifiées**, le paquet n'est pas automatiquement reconstruit. Après avoir fait ces changements, **la reconstruction de ce paquet est souvent suffisante**.
- **Quand un changement au squelette de système de fichiers racine est fait, une reconstruction complète est nécessaire**. Cependant, lorsque **des modifications du système de fichiers racine sont faites dans le dossier de sortie (output/target) ou dans le dossier overlay, dans un script de post-construction ou dans un script de post-image, il n'y a pas besoin d'une reconstruction complète**.

Pour en savoir plus sur Buildroot et son utilisation, reportez vous au manuel :

http://buildroot.uclibc.org/downloads/manual/manual.html

5 Raspberry pi - le matériel

- Quel est la référence du processeur qui équipe votre carte Raspberry Pi 2 ?
- Quel est son type d'architecture ? Combien de cœurs possède-t-il ?
- Quel est la quantité de RAM disponible ?
- La mémoire est-elle gérée par une MMU ?
- Quel est le ratio entre la mémoire CPU/GPU par défaut ?
- Quels sont les bus de communication disponibles ?
- Quelle version de VFP (Vector Foating Point) support-il ?

6 Buildroot pour Raspberry Pi 2

6.1 Téléchargement

Nous allons créer ici un système linux complet pour Raspberry Pi 2 à partir d'une version de Buildroot native qu'il faudra configurer :

https://buildroot.org/download.html (NE PAS TELECHARGER ! C'est déjà fait. Lire la suite.)

6.2 Construction d'un système de base

La compilation d'un système avec les options par défaut peut être particulièrement longue, aussi, une pré-compilation a déjà été effectuée dans le dossier **~/Documents/TP-Buildroot/buildroot-2017**.

- Prenez connaissance du fichier **README**, quelles sont les actions à mener pour construire un système ?
- Supprimez le fichier **.config** dans le dossier **buildroot-2017**.
- Consultez le menu de configuration de buildroot, notamment **Target options**.
- Prenez connaissance du fichier de définition de la configuration pour une cible Raspberry Pi 2. Les fichiers de configuration standard des différentes cibles supportées par Buildroot se trouvent dans le dossier **configs**. Analysez le contenu de ce fichier et identifiez les principaux changements qui seront effectués par l'application de configuration, notamment par rapport aux spécificités du matériel.
- Appliquez la commande **make raspberrypi2_defconfig**. Observez la création du fichier **.config**.
- Consultez le fichier **readme.txt** du dossier **board/raspberrypi2**. Déployez le système obtenu sur votre carte micro-SD. Quelle est la taille du noyau et du système de fichier (en Mo). Décrivez les différentes étapes à mener.
- Testez votre système et commentez les éléments suivant:
    - Estimez la durée de boot
    - Bannière d'accueil, mot de passe root
    - Allure du prompt, disposition du clavier
    - Connexion réseau, connexion à distance par ssh
    - Utilisation des GPIO
    - Identifiez la version du noyau linux installé.
- Consultez le script **/etc/init.d/rcS** et expliquez la procédure de démarrage du système.
    - Consultez et expliquez le fonctionnement des scripts **/etc/network/interfaces** et **/etc/init.d/Sxxnetwork** pour que l'interface **eth0** soit activée et recherche ses paramètres IP automatiquement.
      Remarque : votre système minimum ne dispose que de **vi** comme éditeur !

6.3 Amélioration du système

6.3.1 Clavier AZERTY

En connexion directe avec un clavier USB et un écran HDMI, le clavier est reconnu suivant une organisation Qwerty et non Azerty.

Pour configurer le clavier correctement, il faut appeler, au sein d'un script de démarrage, la commande **loadkmap** de **Busybox** en lui envoyant sur son entrée standard le contenu d'un fichier de configuration. On obtient ce fichier simplement en invoquant l'applet **dumpkmap** de **Busybox** après l'avoir recompilé sur un système où le clavier est configuré correctement comme, par exemple, votre PC de développement :

```
$ sudo busybox dumpkmap > azerty.kmap
```

Le fichier **azerty.kmap** obtenu doit ensuite être transféré dans le dossier **/etc** de la cible et invoqué dans un script de démarrage. Vous pourrez être amené à modifier la taille de la partition du système de fichier et relancer une construction (**Filesystem Image -> exact size**).

- Créez le fichier **azerty.kmap** sur votre hôte et transférez-le sur la cible.
- Créez dans le dossier **/etc/init.d** de la carte micro-SD un script de démarrage **S10keyboard** pour charger le gestionnaire de clavier à l'aide de la commande :

  ```
  loadkmap < /etc/azerty.kmap
  ```

- Redémarrer la cible et testez la configuration du clavier.
- Expliquez le contenu de votre script et son fonctionnement lors du processus de démarrage.

6.3.2 Prompt du shell global

Le système minimal présente un shell minimaliste dans lequel il n'est pas aisé de savoir où l'on se trouve dans le système de fichier et où l'utilisateur connecté n'est pas identifié clairement. On sait juste s'il s'agit ou non de root (**$** ou **#**).

Pour configurer le prompt du shell afin qu'il indique clairement l'utilisateur connecté et le dossier courant, il faut modifier le fichier **/etc/profile**. Le prompt du shell d'un terminal y est représenté par la variable **PSn** ou n est le numéro du terminal. Ici, nous n'avons qu'un seul terminal configuré : **PS1**

Pour obtenir un prompt de type **user@hostname : directory**

```
...
if [ "$PS1" ]; then
    export PS1="\u@\h:\w "
...
      if [ "`id -u`" -eq 0 ]; then
            export PS1=${PS1}"# "
      else
            export PS1=${PS1}"$ "
      fi
```

- Modifiez le fichier **/etc/profile** comme ci-dessus.
- Redémarrer la cible et observez l'allure du prompt quand vous changez de dossier.
- Expliquez le test `if [ "`id -u`" -eq 0 ]; then ...` du script.

6.3.3 Accès à distance

La prise en main à distance d'un système est en général assuré au moyen d'un serveur ssh. Vous pourrez utiliser au choix le package dropbear ou openSSH. Nous utiliserons ici openSSH :

make menuconfig -> Target packages -> Networking applications -> openssh

Dans les deux cas, il faut au préalable sécuriser l'accès local au système en créant un mot de passe pour root et un au moins un utilisateur local.

- Sécurisez l'accès au système en attribuant un mot de passe à root :
  make menuconfig -> System configuration -> Root password
- La création de comptes d'utilisateurs locaux se fait au moyen d'un fichier texte qui répond à la syntaxe de Makeusers définie dans le chapitre 24 de la documentation de Buildroot :

  https://buildroot.org/downloads/manual/manual.html

  Créez dans le **dossier board/raspberrypi2** un fichier adduser.txt contenant la description du compte utilisateur suivant :

  - **username** : admin
  - **uid** : Laisser faire Buildroot
  - **group** : admin
  - **gid** : laisser faire Buildroot
  - **password** : admin (sera encoder avec MD5)
  - **home** : /home/admin
  - **shell** : /bin/sh
  - **groups** : aucun
  - **comment** : Admin user
- Indiquez à Buildroot la présence de ce fichier pour créer notre utilisateur conformément au paragraphe **9.6. Adding custom user accounts** de la documentation de Buildroot
- Modifiez la taille exact du système de fichier à 100Mo (**Filesystem Image -> exact size**)
- Relancez le processus de construction (il sera partiel et ne prendra en compte que les changements à effectuer sur le système de fichiers rootfs.ext)
- Déployez le système obtenu sur votre carte micro-SD.
- Redémarrez la cible et observez les modifications réalisées (le prompt et le clavier ?).
- Assurez vous de la nécessité de fournir le bon mot de passe root pour accéder au système.
- Accédez à la cible depuis votre PC hôte via une session ssh. Assurez-vous que root ne puisse pas y accéder directement.
- Expliquez les différentes étapes que vous avez mené.
- Expliquez pourquoi root ne devrait jamais pouvoir ouvrir une connexion distante directement.

6.3.4 Persistance des modifications sur rootfs

Les modifications réalisées directement sur cible ou sur la carte micro-SD à partir de l'hôte fonctionnent très bien pour un système unique et final. Cependant, lors la prochaine reconstruction du système, ces changements auront disparu car ils ne sont pas présents dans la configuration de buildroot.

Il est important que le processus de construction reste entièrement reproductible, si l'on veut s'assurer que la prochaine version inclura les configurations personnalisées.

Pour ce faire, le plus simple est d'utiliser le mécanisme de recouvrement de rootfs de Buildroot (overlay mechanism). Cette substitution de fichiers est spécifique au projet en cours. Il faut donc créer un répertoire personnalisé pour ce projet dans le source de buildroot pour la cible visée :

board/<manufacturer>/<boardname>/fs-overlay/

Dans la config de buildroot (**buildroot menuconfig**), il faut spécifier ce chemin dans l'option **Root filesystem overlay directories** du menu **System configuration**.

Il faut ensuite recréer l'architecture des dossiers contenant les scripts à personnaliser dans le dossier **fs-overlay** :

```
marco@marco-CERI:~/Documents/buildroot/buildroot-2017.11/board/raspberrypi2$ tre
e fs-overlay/
fs-overlay/
└── etc
    ├── azerty.kmap
    ├── init.d
    │   └── S10keyboard
    └── profile
```

- Créez le dossier de recouvrement **fs-overlay** et copiez-y les scripts précédents assurant la bonne configuration du clavier et du prompt.
- Indiquez la présence de ce dossier à buildroot.
- Ajoutez au système un nom d'hôte et une bannière d'accueil.
  make menuconfig -> System configuration -> System hostname
  -> System banner
- Relancez le processus de construction.
- Remplacez le système de fichier racine sur la carte micro-SD.
- Redémarrez la cible et observez les modifications réalisées.

7 Projet : Ajout du code métier

7.1 Objectif

Le but de ce mini-projet est de réaliser un télémètre à ultrason avec indicateur local de dépassement et une IHM déportée.

7.2 Spécifications

- L'indicateur local de dépassement de distance est une LED connectée à une broche GPIO ou la LED verte déjà présente sur la carte et qui sert normalement à visualiser les accès à la carte micro-SD.
- Le télémètre est un capteur ultrasonic ranger V3 de chez Seeed : http://wiki.seeed.cc/Ultra_Sonic_range_measurement_module/

- L'interface graphique est un site web permettant à l'utilisateur de lire les distances mesurées.
- Le système doit pouvoir être accédé à distance au moyen d'une connexion sécurisé.

7.3 Indications

7.3.1 Temps de construction

Attention au temps de construction de votre système qui peut être très long suivant les modifications que vous souhaitez faire. Reportez vous au paragraphe 3.7 du présent document pour plus de renseignement sur le processus de construction total ou partiel (à privilégier !).

7.3.2 LED verte de la carte Raspberry Pi

Le support kernel pour les LED est intégré lors de la compilation du noyau (**make linux-menuconfig**) en activant l'option « LED **Support for GPIO connected LEDs** » dans le sous-menu **LED support** du menu **Device Drivers**. Avec le Raspberry Pi, cette option se traduit par l'apparition d'un sous-répertoire **led0** dans **/sys/class/leds**.

Dans le même sous-menu de configuration du kernel, on peut choisir des déclencheurs (triggers) – c'est-à-dire des heuristiques commandant l'allumage ou l'extinction des LED. Le trigger sera sélectionné en écrivant son nom dans **/sys/class/leds/led0/trigger**.

Plusieurs heuristiques sont disponibles, en voici quelques-unes :

- **none** : la LED est pilotée manuellement en écrivant **0** ou **1** dans **/sys/class/leds/led0/brightness**.
- **timer** : (module ledtrig-timer.ko compilé grâce à l'option « **LED Timer Trigger** » lors de la configuration du noyau) la LED clignote suivant un cycle dont on peut contrôler les durées d'allumage et d'extinction en indiquant les valeurs en millisecondes dans les fichiers **/sys/class/leds/led0/delay_on et /sys/class/leds/led0/delay_off**.
- **heartbeat** : (module ledtrig-heartbeat.ko dépendant de l'option « LED Heartbeat Trigger ») pour ce trigger, la LED simule un battement de cœur avec une fréquence qui dépend de la charge système (c'est celui que nous choisirons plus bas).
- **backlight** : (option « LED Backlight Trigger ») : utilisé lorsque la LED sert de rétro-éclairage pour un écran.
- **mmc0** : intégré directement dans le driver **mmc-core**, ce trigger allume la LED lors des accès à la carte SD.

7.3.3 Connexion du capteur à ultrason

Il se connecte à une broche GPIO :

- VCC : 3.3V
- SIG : GPIO21
- GND : GND

7.3.4 Serveur web et prise en main à distance sécurisée

Ces deux fonctionnalités peuvent être réalisées par l'ajout de packages dans buildroot.

Le serveur web pourrait être lighttpd ou nginx auquel on pourrait ajouter php.

make menuconfig -> Target packages -> Networking applications -> lighttpd ou nginx
-> openssh (ou dropbear)
-> Interpreter languages and scripting -> php

Il peut également être constitué d'un serveur websockets comme l'exécutable websocketd disponible dans le dossier **~/Documents/TP-Buildroot/RaspberryPing**

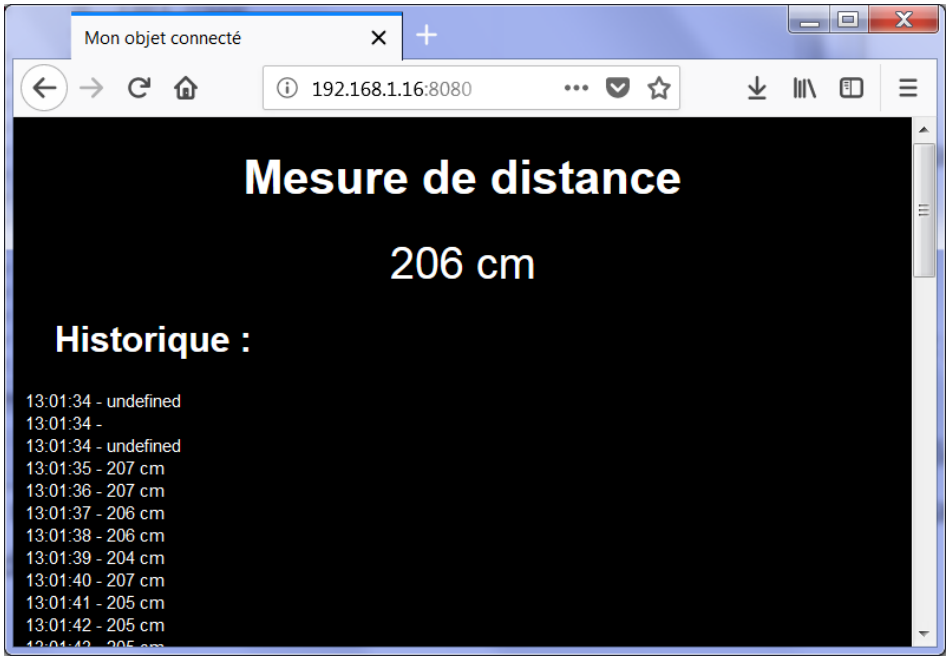
La prise en main à distance doit être réalisée au moyen d'un serveur ssh.

7.3.5 Fichiers du projet

Les fichiers du projet sont disponibles sur l'ENT ou dans le dossier **~/Documents/TP-Buildroot/RaspberryPing**. Ils doivent être cross-compilés sur le PC hôte en utilisant la croostool chain fabriqué par buildroot pour le Raspberry Pi et localisée ici : **/output/host/usr/bin/**

- Exportez le chemin d'accès à la chaine de cross-compilation.
- Compilez les fichiers du projets pour obtenir un exécutable que vous nommerez **ping**.
- Transférez l'exécutable **ping** obtenu sur le Raspberry et exécutez le.
- Transférez le serveur websocket websocketd et la page index.html associée sur le Raspberry et excutez ping pour rediriger sa sortie vers les sockets clients. Testez dans un navigateur l'accès aux mesures de distances.

./websocketd --port=8080 --staticdir=. ./ping



- Modifiez éventuellement le programme pour que la led verte présente sur le Raspberry simule un battement de cœur (mode **hearthbeat**).

7.3.6 Construction de l'image finale

- Construisez une image intégrant toutes les fonctionnalités nécessaires pour rendre opérationnel le système dès son déploiement. Pour cela, il vous faudra ajouter à votre configuration Buildroot précédente les exécutables **ping** et **websoscketd**, la page web **index.html** et écrire un **script de démarrage automatique** de l'ensemble. Tous ces fichiers sont à ajouter au mécanisme de substitution de fichier **fs-overlay**.
- Relancez le processus de construction.
- Remplacez le système de fichier racine sur la carte micro-SD.
- Redémarrez la cible et faites constater le fonctionnement immédiat du système.

8 Références web

8.1 Buildroot

- http://buildroot.uclibc.org/
- http://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-155/Linux-from-scratch-Construire-une-chaine-de-compilation
- http://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-155/Linux-from-scratch-Construire-un-systeme-complet
- http://connect.ed-diamond.com/GNU-Linux-Magazine/GLMF-158/Raspberry-Pi-from-scratch-3
- http://linux.developpez.com/tutoriels/embarque-buildroot/
- http://free-electrons.com/fr/formation/buildroot/
- https://github.com/gamaral/rpi-buildroot

8.2 Raspberry Pi

- https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
- http://www.adafruit.com/pdfs/raspberrypi2modelb.pdf
- https://learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-2-model-b.pdf
- https://en.wikipedia.org/wiki/ARM_Cortex-A7

8.3 Linux

- https://www.kernel.org/doc/man-pages/
- https://github.com/raspberrypi/linux.git
- https://github.com/torvalds/linux

8.4 Buzybox

- http://www.busybox.net/

8.5 vi

- http://www.linux-france.org/prj/support/outils/vi.html
- http://wiki.linux-france.org/wiki/Utilisation_de_vi