Image Viewer

Objectif :
Les objectifs de ce travail pratique sont les suivants :

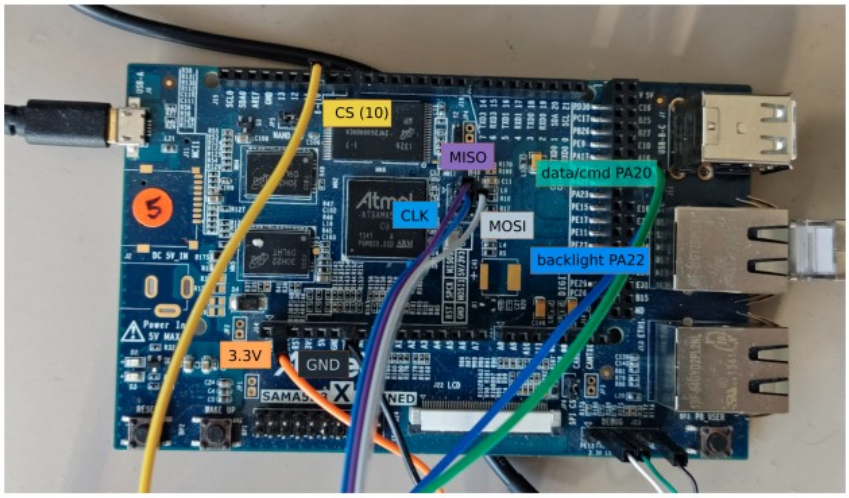
— Accès à l'écran LCD de la carte MyLab2 via la carte Sama5D3 Xplained sous Linux, en particulier être capable d'afficher des images au format PPM sur l'écran LCD.

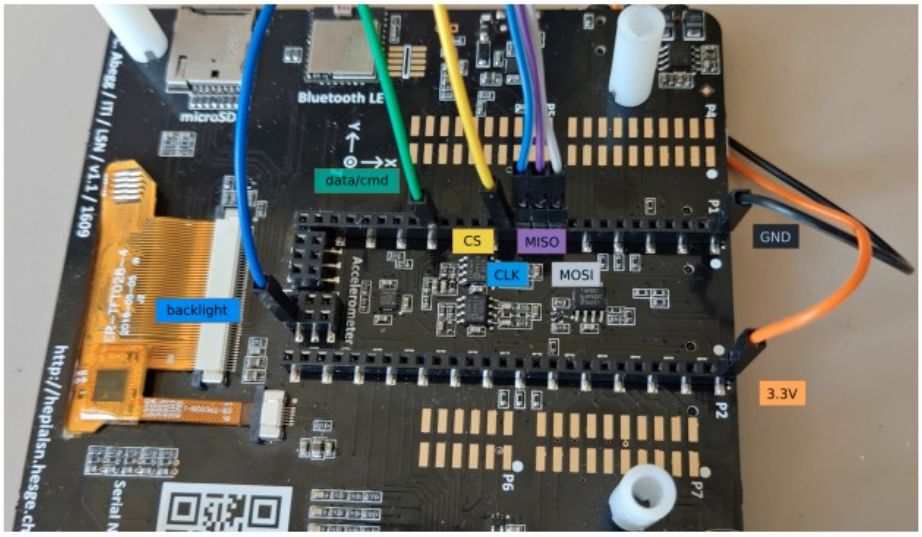
— Développement d'une API REST s'exécutant sur la carte Sama5D3 (serveur backend) offrant les fonctionalités suivantes :
- — ajouter une image sur le serveur
- — supprimer une image du serveur
- — lister les images se trouvant sur le serveur

— afficher une image présente sur le serveur sur l'écran LCD de la carte MyLab2.

— Integration de l'application dans Buildroot, ce avec toutes les dépendences nécessaires.

Cablage de la carte :

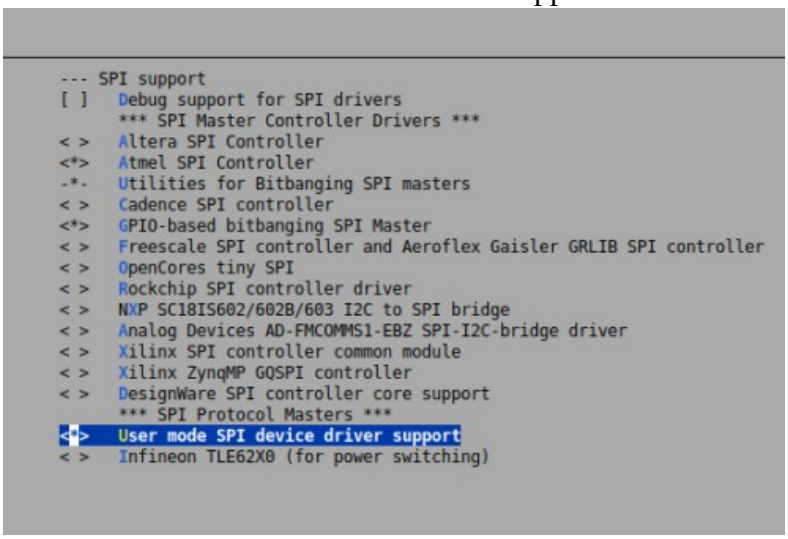




Ajout du support SPI :

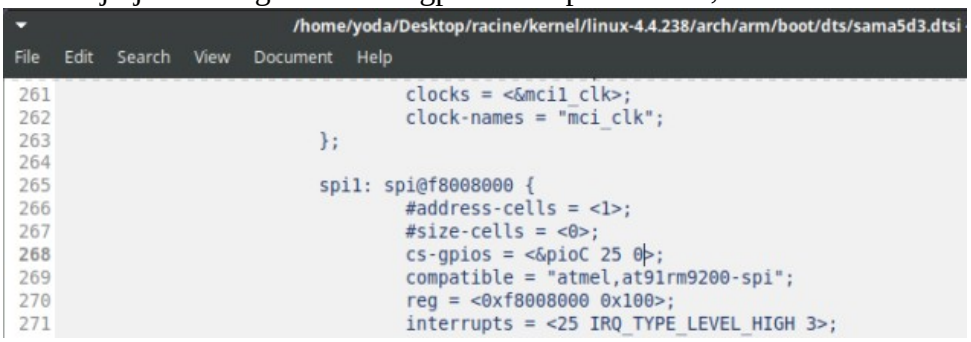
Je vais dans le dossier du kernel, et je lance la commande :
make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf-gcc ARCH=arm menuconfig

J'active User mode SPI device driver support : dans Device Driver → SPI Support

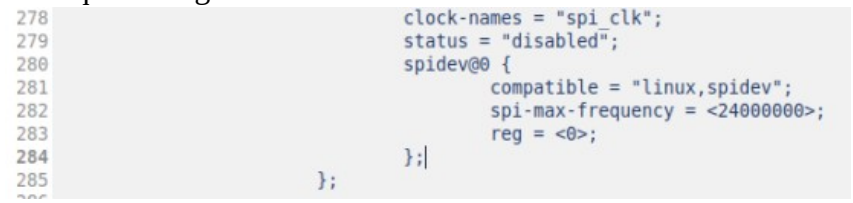


→ ici

Ensuite j'ajoute la ligne 268 : cs-gpios = <&pioC 25 0>;

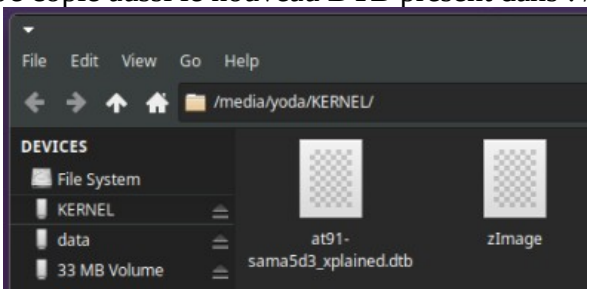


Ainsi que ces lignes :

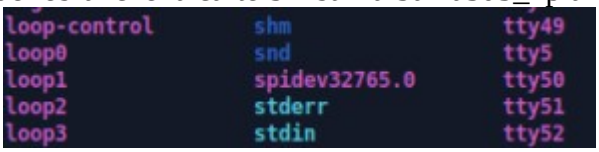


Puis je recompile le kernel avec la commande :
> make CROSS_COMPILE=arm-buildroot-linux-uclibcgnueabihf- ARCH=arm -j12

Ensuite je copie la nouvelle zImage qui se trouve dans : arch/arm/boot/zImage sur la carte SD.
Je copie aussi le nouveau DTB présent dans : /arch/arm/boot/dts/at91-sama5d3_xplained.dtb



Je rebranche la carte SD sur la Sama5d3_xplained et essaie de démarré.



Maintenant un device spidev apparaît.

Mise en place de l'interface Web :

je commence par ajouter python3 a buildroot :
> make clean
> make menuconfig (Target Packages → Interpreter Language… → Python3 → External Python-Flask module, et requests)
> make -j12

Ensuite je décompresse l'archive dans nfsroot, et j'exécute :
> sudo chown root * -R

Je remarque qu'il me faut aussi le package convert sur le système embarqué, donc je l'installe :
> make clean
> make menuconfig
cocher les case suivante :
1 : Target Packages → Graphic libraries → imagemagick
2 : Target Packages → Libraries → Graphics → libpng
3 : Target Packages → Libraries → Graphics → jpeg support
> make -j12

Encore une fois je décompresse l'archive dans nfsroot, et j'exécute :
> sudo chown root * -R

Ces deux étape peuvent être faite en une seule fois…

Développement de l'API et de l'affichage :
L'API prend en charge ces requêtes :

`/upload`	méthode POST pour d'uploader une image sur le serveur
`/download/<image>`	méthode GET pour télécharger une image depuis le serveur
`/list`	médhode GET pour lister les images disponibles sur le serveur
`/delete/<image>`	méthode DELETE pour supprimer une image du serveur
`/display/<image>`	méthode GET affichant l'image `image` sur l'écran LCD

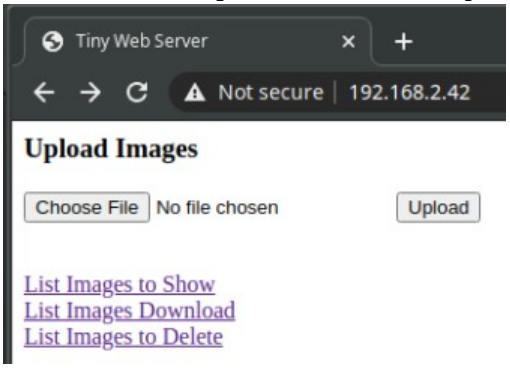
+ une requête /displayRandom qui affiche une image uploader aléatoirement.
Toutes les requête retourne un json, contenant soit «OK», soit «ERROR»

Ensuite l'affichage se fait en passant par l'exécutable show_image. Le programme fonctionne comme un serveur. Il est lancée au démarrage de la carte, ensuite il attend le signal USER1. Lorsqu'il reçoit le signal USER1, il affiche l'image contenu dans .tmp/to_show.ppm puis la supprime. Ce qui signifie que quand il n'y a pas d'image to_show dans .tmp, le programme est disponible, sinon il ne l'est pas.

Donc pour géré les problème de concurrence a l'affichage, il faut faire attention de tester si le fichier to_show.ppm est présent, de manière atomic. Si il est présent la requête renvoie une erreur, et l'image n'est pas affiché. Sinon, le programme est disponible pour afficher une image.
Voici la fonction display dans api.py.

```
69      try:
70          #atomic test si le fichier existe en le créant
71          os.open("/API/.tmp/to_show.ppm", os.O_CREAT | os.O_EXCL)
72          shutil.copyfile("/API/upload/"+name, "/API/.tmp/"+name)#copie le fichier pas atomic !!!
73          #rename est atomic, et vas effacer le fichier crée par le test au dessus
74          #en la remplacent par la bonne image a afficher
75          os.rename("/API/.tmp/"+name,"/API/.tmp/to_show.ppm")
76          #un fois que tout est bon, envoie le signal pour afficher l'image
77          os.kill(get_pid("show_image"), signal.SIGUSR1)
78      except Exception as e:
79          #os.remove(".tmp/"+name)
80          return jsonify({'ERROR': 'Cant show image'}), 500
81      return jsonify({'OK':'Image '+name+" on screen"}), 200
```

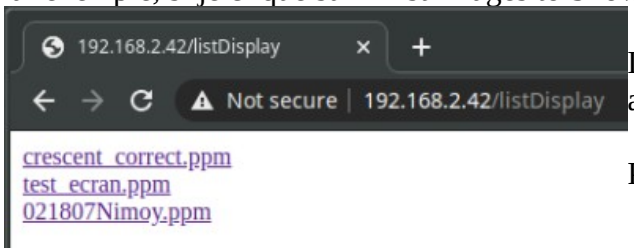
J'ai aussi mis en place un frontend, qui utilise l'API, voici la page web :



→ Permet d'uploader des fichier.

→ Permet de faire des opérations sur sur les image

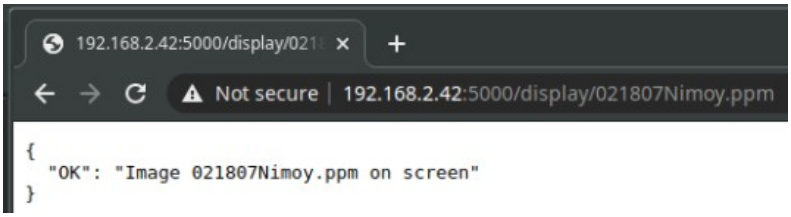
Par exemple, si je clique sur «List Images to Show» :



Le serveur affiche les image qui peuvent être afficher sur le serveur.

Pour les afficher, il suffit de cliquer dessus.

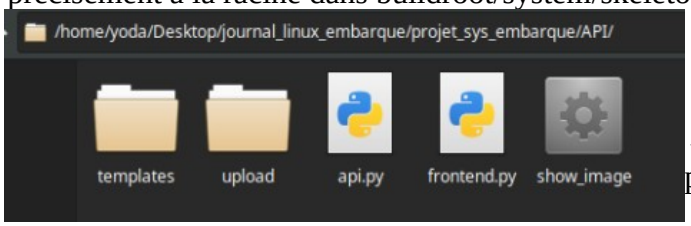
Par exemple en cliquant sur 021807Nimoy.ppm :



→ L'image est affiché

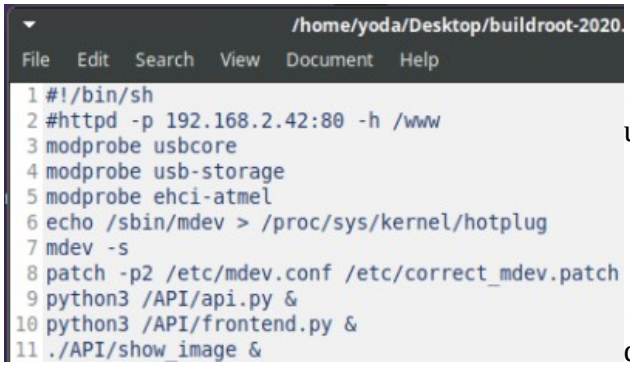
Ajout a buildroot :

Je commence par copier le dossier API qui contient les script python dans l'overlay : precisement a la racine dans buildroot/system/skeleton/



→ il faudrait ajouter show_image comme package, mais ici je vais le laisser comme ça.

et je modifie le fichier S01rcS dans l'overlay :buildroot/system/skeleton/etc/init.d/



→ je met en commentaire httpd car il n'est plus utilisé

→ je lance l'API, le frontend, et le logiciel d'affichage.

Il ne reste plus qu'as tester le rootfs :
> make clean
Normalement pas besoin de menuconfig pour réactivé les ancien package...
> make -j12

Encore une fois je décompresse l'archive dans nfsroot, et j'exécute :
> sudo chown root * -R

Tout fonctionne.