

Y.N.O.V.

PROJET DE MASTER EN PARTENARIAT AVEC THALÈS

Intégration d'un driver sur une OpenRex Basic

Auteurs :

Alain AIT-ALI

Martin LAPORTE

Clément AILLOUD

Romain PETIT

Superviseurs :

David COUÉ

*Rapport hebdomadaire présentant l'avancement du projet sur l'intégration d'un driver de la
Raspi Cam v2 sur un imx6 OpenRex Basic*

au sein du

Département Aéronautique & Systèmes Embarqués



20 décembre 2017

Table des matières

1	Organisation de l'équipe et planning	1
1.1	Organisation de l'équipe	1
1.2	Planning	1
2	Solution n° 1	2
3	Solution n° 2	4
4	Solution n° 3	6
4.1	Avancement	6
4.2	Source des patches	6
4.3	Ajout de paquets	6
5	État de l'art de la récupération du flux vidéo	8
5.1	Webcam USB	8
5.2	Problèmes	9
5.3	Point d'amélioration	9
5.4	Sources	9
6	Conclusion	10

Table des figures

1.1	Avancement du projet actuel	1
3.1	Arborescncce des fichiers	4
3.2	Vérification de la présence du driver	4
3.3	Chargement du module	5
3.4	Chargement du module avec l'adresse I2C	5
5.1	Port USB de la webcam USB	8
5.2	Vérification du port et identifiant de la webcam USB	8
5.3	Liaison entre la webcam USB et son driver	8
5.4	Capture du flux vidéo de la webcam USB	9
5.5	LED verte allumée lors de l'envoi de la commande	9

Chapitre 1

Organisation de l'équipe et planning

1.1 Organisation de l'équipe

Après notre entrevue, nous avons décidé de nous séparer en deux groupes un qui partirait du haut niveau et un autre qui partirait du bas niveau le but de cette méthode est de pouvoir mieux appréhender les différents problèmes et axes de développement. Le binôme Clément & Romain recherche à faire le lien depuis les couches applicatives vers le kernel. Le binôme Alan & Martin au contraire cherche à établir en priorité les couches basses pour ensuite les rendre compatible avec le kernel. Pour résumer les deux binômes n'ont pas le même point de départ tout en ayant la même version du kernel.

Clément et Romain sont donc chargés de l'utilisation de Gstreamer et de la couche V4L alors que Martin et Alan sont chargés de rendre les drivers compatibles avec notre système.

1.2 Planning

Nous vous avons ici présenté un planning des taches effectuer durant ces 8 derniers jours, les noms attribués à chaque tache seront développées dans la suite du rapport.

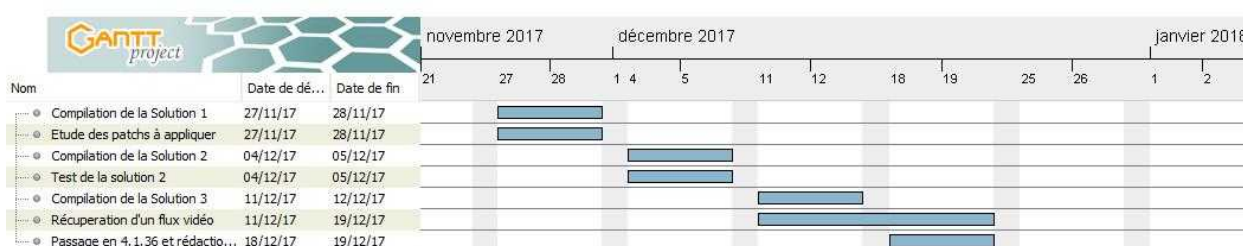


FIGURE 1.1 – Avancement du projet actuel

Chapitre 2

Solution n° 1

Nous avons récupéré le code du driver à exploiter dans les premiers mails. Les deux fichiers imx219.c et camera.h sont indiqués comme « Linux V4L2 Driver » et sont téléchargeables à l'adresse suivante :

<https://www.raspberrypi.org/forums/viewtopic.php?f=43&t=162722>

Ce driver lie un processeur Allwinner A80 (ARM Cortex-A15/A7) à un composant pilote vidéo imx219. Dans un premier temps nous nous doutions qu'il y aurait des erreurs de compilation donc nous avons préféré décorréliser les erreurs de compilation des erreurs Yocto. Nous avons donc essayé de compiler ce fichier depuis une cross-toolchain générée par Yocto en utilisant la variable CC qui configure le cross compilateur. Cette variable est affectée lors du sourcing du sdk généré par :

```
1 #génération du sdk
2 bitbake openrexpica-base-image -c populate_sdk
3 # environnement du sdk
4 source
5   /opt/poky/2.0.3/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi
6 # aperçu de CC
7 $CC imx219.c
```

Cette compilation fait appel à des bibliothèques contenues dans l'arborescence de compilation de différentes recettes.

```
1 BUILDDIR=/home/diag/workspaceThales/Yocto/fsl-community-bsp/build-imx6rex.com
2 IMX6S_DIR=$BUILDDIR/tmp/work/imx6s-openrex-poky-linux-gnueabi
3 #libs linux/*.h
4 DIRLINUX=$IMX6S_DIR/linux-openrex/3.14-r0/git/include
5 DIRLINUX2=$IMX6S_DIR/u-boot-openrex/v2015.10+gitAUTOINC+7d8ddd7de7-r0/git
6 /include
7 DIRLINUX3=$IMX6S_DIR/core-image-minimal/1.0-r0/sdk/image/opt/poky/2.0.3
8 /sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi/usr/include
9 #libs asm/*.h
10 DIRASM=$BUILDDIR/tmp/work/x86_64-nativesdk-pokysdk-linux
11 /nativesdk-linux-libc-headers/4.1-r0/linux-4.1/arch/arm/include/
12 DIRASM2=$IMX6S_DIR/linux-openrex/3.14-r0/build/arch/arm/include/generated
13 DIRASM3=$IMX6S_DIR/u-boot-openrex/v2015.10+gitAUTOINC+7d8ddd7de7-r0/git
14 /arch/arm/include/
15 #nouvelle commande de compilation "out of tree"
16 $CC imx219.c -I$DIRLINUX -I$DIRASM -I$DIRASM2 -I$DIRLINUX2
17 -I$DIRLINUX3 -I$DIRASM3
```

Nous avons inclus 6 dossiers de bibliothèques en option puis les nouvelles dépendances étaient inexistantes dans le dossier contenant l'arborescence de compilation donc nous nous sommes mis à chercher cherché une autre solution.

Après avoir parlé à notre professeur de Linux embarqué des soucis de compilation que nous rencontrions nous avons compilé le driver en ajoutant ses sources dans l'arborescence (in-tree).

Cependant certains fichiers n'existent pas dans le kernel 3.14 nous sommes actuellement en-train de d'ajouter le patch au kernel 4.1.38.

Chapitre 3

Solution n° 2

Nous avons trouvé un driver `imx219.c` compatible avec une version 3.14 du kernel soit version Jethro de Yocto. Disponible à cette adresse :

https://chromium.googlesource.com/chromiumos/third_party/kernel/+factory-ryu-6486.14.B-chromeos-3.14/drivers/media/i2c/soc_camera/imx219.c

Le driver `imx219` contient toutes les fonctions permettant la configuration de l'`imx219` (on, off, gain, couleurs., taille de l'affichage..) via le driver `i2c` et le driver `v4l`. Lors du chargement du module, l'`imx219` est automatiquement configuré par `v4l`. Cette configuration permet de modifier certains paramètres du flux vidéo à l'aide des fonctions contenues dans le driver `imx219`. Le flux peut être ensuite récupéré avec les commandes de Gstreamer ou de `v4l-utils`.

Gstreamer permettrait de streamer le flux sur le réseau à travers le protocole UDP alors que `v4l-utils` permettrait plutôt d'enregistrer le flux vidéo.

Nous avons commencé par compiler le driver sans l'intégrer directement au système (linux-openrexx) par l'utilisation de patches mais en réalisant une recette qui compilera le fichier.

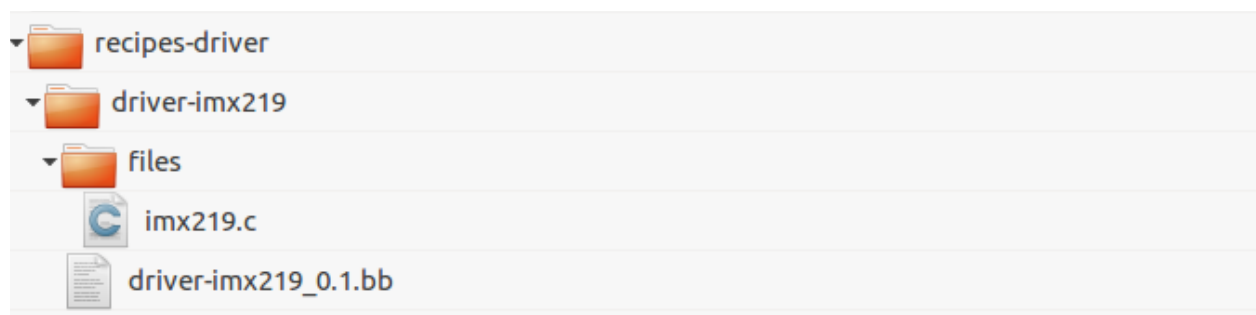


FIGURE 3.1 – Arborescne des fichiers

Extrait de la recette : Elle permet de récupérer le fichier `imx219.c` de le compiler par la cross tool-chain et de l'installer.

```
root@openrexpica:~# find / -name imx219.ko
/lib/modules/3.14.61-yocto+g336bc38/extra/imx219.ko
root@openrexpica:~# ls /lib/modules/3.14.61-yocto+g336bc38/extra/
galcore.ko  imx219.ko
```

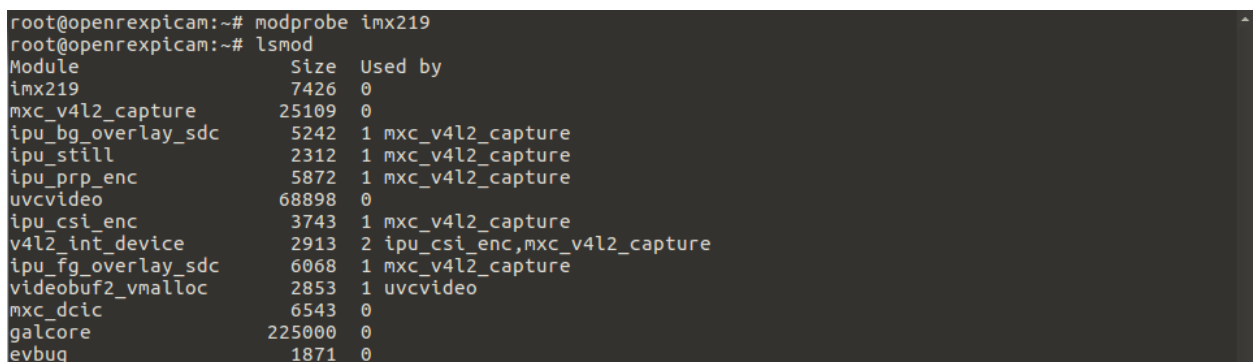
FIGURE 3.2 – Vérification de la présence du driver

```

1 SUMMARY = "driver_imx219"
2 SECTION = "examples"
3 LICENSE = "MIT"
4 LIC_FILES_CHKSUM =
5     "file://${COMMON_LICENSE_DIR}/MIT;md5=c2b5f7071fdde268bf46ace1546f3c4b"
6 SRC_URI = "file://imx219.c"
7 S = "${WORKDIR}"
8
9 do_compile() {
10     ${CC} imx219.c -o imx219
11 }
12
13 do_install() {
14     install -d ${D}${bindir}
15     install -m 0755 imx219 ${D}${bindir}
16 }

```

Chargement du module imx219 :



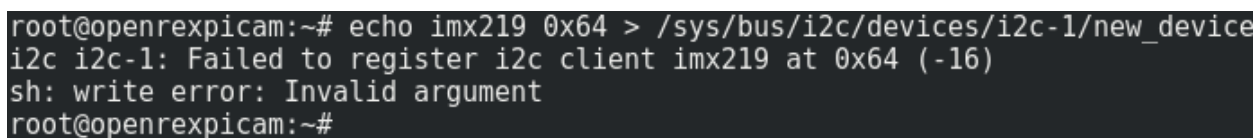
```

root@openrexpica:~# modprobe imx219
root@openrexpica:~# lsmod
Module                  Size  Used by
imx219                  7426  0
mxc_v4l2_capture        25109  0
ipu_bg_overlay_sdc     5242  1 mxc_v4l2_capture
ipu_still              2312  1 mxc_v4l2_capture
ipu_prp_enc            5872  1 mxc_v4l2_capture
uvcvideo               68898  0
ipu_csi_enc            3743  1 mxc_v4l2_capture
v4l2_int_device         2913  2 ipu_csi_enc,mxc_v4l2_capture
ipu_fg_overlay_sdc     6068  1 mxc_v4l2_capture
videobuf2_vmalloc      2853  1 uvcvideo
mxc_dcic               6543  0
galCore                225000  0
evbug                  1871  0

```

FIGURE 3.3 – Chargement du module

Comme nous pouvons le voir le driver n'est pas encore utilisé par v4l (Used by 0). Chargement du module à l'adresse I2C de l'imx219 :



```

root@openrexpica:~# echo imx219 0x64 > /sys/bus/i2c/devices/i2c-1/new_device
i2c i2c-1: Failed to register i2c client imx219 at 0x64 (-16)
sh: write error: Invalid argument
root@openrexpica:~#

```

FIGURE 3.4 – Chargement du module avec l'adresse I2C

Nous arrivons à une erreur sur le bus i2c sur laquelle nous travaillons actuellement. Récupération du flux vidéo en local via Gstreamer :

gst-launch-1.0 v4l2src device="/dev/videoX"! video/x-raw,width=640,height=480! autovideosink

X' étant le numéro correspondant au numéro du driver vidéo du port CSI-2

Nous sommes actuellement entrain de chercher des pistes sur l'utilisation des commandes gst-launch-1.0 et les fichiers de configurations v4l2src la suite de cette partie sera développer à la suite de ce rapport.

Chapitre 4

Solution n° 3

4.1 Avancement

Nous avons récemment trouvé un driver `imx219.c` qui ne contient aucune librairie associées et qui fonctionne sur une plateforme hummingboard (SOC i.MX6).

Les sources étant sous forme de patch nous allons cette fois directement l'intégrer à notre système.

Le driver est composé de deux patches, le premier patch comporte tous les éléments proches du kernel on retrouve :

Kconfig : permet d'ajouter l'`imx219` au `menuconfig`, cela va nous permettre de demander la compilation du driver. Lorsque l'on modifie le `menuconfig` on récupère un `defconfig` que l'on ajoute en a notre recette.

Makefile : permet d'ajouter la compilation de driver (elle sera activée ou non par le `defconfig`)

`imx219.c` : driver qui contient l'ensemble des configurations spécifiques à la caméra.

Le deuxième patch permet d'ajouter les configurations utiles pour faire l'interface entre le driver et l'utilisateur soit le `device-tree`.

Problèmes rencontrés :

- Des librairies système sont inexistantes, nous les avons donc identifiées et ajoutées au système.
- Les versions des librairies existantes ne sont pas compatibles avec notre fichier `imx219.c`. Nous utilisons une version de kernel 3.14 ou bien la version nécessaire est bien plus récente nous l'estimons à 4.12. Pour s'approcher de la versions 4.12 nous sommes actuellement en-train d'appliquer les patches sur une version plus récente soit 4.1.36 (Krogoth sous Yocto).

Vous trouverez ici un lien qui contient l'ensemble des patches appliqués sur la version 3.14 du kernel :

<https://github.com/Alanaitali/meta-openrexplicam/tree/develop/gladwistor/recipes-kernel/linux>

4.2 Source des patches

<http://git.armlinux.org.uk/cgit/linux-arm.git/commit/?h=csi-v6&id=e3f847cd37b007d55b76282414bfcf>

<http://git.armlinux.org.uk/cgit/linux-arm.git/commit/?h=csi-v6&id=4bd8e1231a2e6eca6a65b565176ea9>

4.3 Ajout de paquets

Sachant que nous allons avoir besoin d'une couche `v4l-utils` et `gstreamer` comme applicatif (user-space), nous avons commencé par essayer de le compiler. Cette compilation n'a abouti ni par compilation in-tree et out-of-tree, ni par une recette Yocto.

Par la suite nous avons ajouté certains paquets à notre image via la variable IMAGE_INSTALL.

```
1 DESCRIPTION = "Basic_image_openrexpica"
2 LICENSE = "MIT"
3
4 inherit core-image
5
6 IMAGE_INSTALL += "\
7     gststreamer\
8     i2c-tools\
9     gststreamer1.0-plugins-imx\
10    gst1.0-fsl-plugin\
11    v4l-utils\
12    "
```

Chapitre 5

État de l'art de la récupération du flux vidéo

5.1 Webcam USB

Afin de mieux comprendre l'utilisation de Video4Linux nous avons effectué plus de recherches sur celui-ci et nous sommes tombés sur un blog expliquant la récupération d'un flux vidéo d'une webcam USB avec V4L2 sur imx6. Nous avons donc essayé de capturer le flux de notre webcam USB dans un premier temps, une fois ceci fonctionnel nous passerons sur le flux vidéo de la Raspi Cam v2. Nous avons récupéré une webcam USB afin de tester les différentes commandes proposées par le blog et essayer de capturer le retour vidéo.

Dans un premier temps il est nécessaire de récupérer sur quel port est connecté la webcam USB, pour cela on utilise la commande : **lsusb -t**

```
root@openrexpica:~# lsusb -t
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ci_hdrc/1p, 480M
   |__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
      |__ Port 2: Dev 3, If 0, Class=Video, Driver=uvcdvideo, 480M
      |__ Port 2: Dev 3, If 1, Class=Video, Driver=uvcdvideo, 480M
```

FIGURE 5.1 – Port USB de la webcam USB

D'après la figure ci-dessus, nous en déduisons que la webcam USB est sur le bus I2C n°1 et l'identifiant du port 1.2.

```
root@openrexpica:~# ls /sys/bus/usb/drivers/usb
1-1      1-1.2  bind    uevent  unbind  usb1
```

FIGURE 5.2 – Vérification du port et identifiant de la webcam USB

Ensuite il faut écrire le numéro du bus et l'identifiant du port dans les fichiers bind et unbind. Le fichier bind permette de « lier » un appareil USB à son driver et ainsi le rendre visible pour le système tandis que le fichier unbind lui détache le driver de son appareil USB pour le « cacher » du système.

```
root@openrexpica:~# echo 1-1.2 > /sys/bus/usb/drivers/usb/unbind
root@openrexpica:~# echo 1-1.2 > /sys/bus/usb/drivers/usb/bind
uvcdvideo: Found UVC 1.00 device Microsoft LifeCam (045e:074a)
input: Microsoft LifeCam as /devices/soc0/soc.0/21000000.aiops-bus/2184200.usb/ci_
hdrc.1/usb1/1-1/1-1.2/1-1.2:1.0/input/input5
```

FIGURE 5.3 – Liaison entre la webcam USB et son driver

On s'aperçoit bien que la webcam USB est bien reconnu par notre système puisque celui-ci nous affiche la référence de la webcam USB comme le montre l'illustration ci-dessus.

```
root@openrexpica:~# gst-launch-1.0 v4l2src device="/dev/video0" ! video/x-raw,w
idth=640,height=480 ! autovideosink
ERROR: v4l2 capture: slave not found!
ERROR: v4l2 capture: slave not found!
Setting pipeline to PAUSED ...
display(/dev/fb0) resolution is (1280x720).
===== OVERLAYSINK: mxc_v4l2_output v4l2_out.25: Bypass IC.
4.0.8 build on Nov 30 2017 12:15:mxc_v4l2_output v4l2_out.25: Bypass IC.
50. =====
display(/dev/fb0) resolution is (1280x720).
display(/dev/fb0) resolution is (1280x720).
Pipeline is live and does not need PREROLL ...
Setting pipeline to PLAYING ...
New clock: GstSystemClock
```

FIGURE 5.4 – Capture du flux vidéo de la webcam USB

La commande `# gst-launch-1.0 v4l2src device="/dev/video0" ! video/x-raw,w`
`idth=640,height=480 ! autovideosink` permet de démarrer un flux vidéo sur notre webcam USB. Nous en sommes sûr puisque lorsque nous lançons la commande celle-ci se fige à « New clock : GstSystemClock », ça se fige exactement au même endroit que si nous effectuons cette commande sur notre propre ordinateur et notre webcam intégré. De plus une petite LED verte s'allume au moment de l'envoi de cette commande.



FIGURE 5.5 – LED verte allumée lors de l'envoi de la commande

5.2 Problèmes

Notre principal problème pour l'instant est de récupérer ce flux vidéo puisqu'il est impossible de l'afficher directement dans le terminal, pour tester cela nous essayons de l'afficher sur nos ordinateurs par le réseau WiFi.

5.3 Point d'amélioration

Dans un premier temps il est nécessaire de mettre l'image à jour pour que l'openrex basic se connecte à un réseau WiFi, qui sera le même que nos ordinateurs, et dans un second temps de récupérer ce flux vidéo par le réseau WiFi sur nos ordinateurs en UDP.

Finalement une fois que nous aurons réussi à afficher le flux vidéo de la webcam USB sur nos ordinateurs nous essayerons d'effectuer le même processus mais pour la Raspi Cam v2, qui est se pilote par le CSI et non l'USB.

5.4 Sources

<http://jas-hacks.blogspot.fr/2014/04/imx6-gstreamer-imx-and-usb-webcam.html>

Chapitre 6

Conclusion

De part l'avancée que nous avons actuellement et l'expérience accumulée au cours du projet ils nous restent plusieurs pistes à explorer. Sans contre indication particulière nous devons être capables d'avoir des résultats peu de temps après la rentrée des vacances de Noël. Nous ne sommes pas en capacité de dire avec précision à quelle date, il sera possible de vous fournir un driver fonctionnel.

Pour tout conseil, piste à suivre ou erreur de notre part n'hésitez pas à revenir vers nous. De plus nous tenons à nous excuser pour le dernier rapport envoyé il n'était manifestement pas du tout adapté à votre demande.