

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



9 janvier 2018

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	3
3.1	Fichiers sources	3
3.2	Travail effectué	3
3.3	L'arborescence des fichiers in-tree	3
3.4	Compilation de driver	4
3.5	Utilisation du driver sur la cible	4
4	Solution n° 3	6
4.1	Contenu des sources	6
4.2	Problèmes rencontrés	6
4.3	lien GIT	6
4.4	Source des patches	7
4.5	Ajout de paquets	7
5	État de l'art de la récupération du flux vidéo	8
5.1	Methodologie par la 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	3
3.2	Vérification de la présence du driver	4
3.3	Chargement du module	4
3.4	Chargement du module avec l'adresse I2C	5
4.1	Emplacement et contenu de la recette de l'openrexpica-base-image	7
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 effectuées durant ces 8 derniers jours, les noms attribués à chaque tache seront développées dans la suite du rapport.

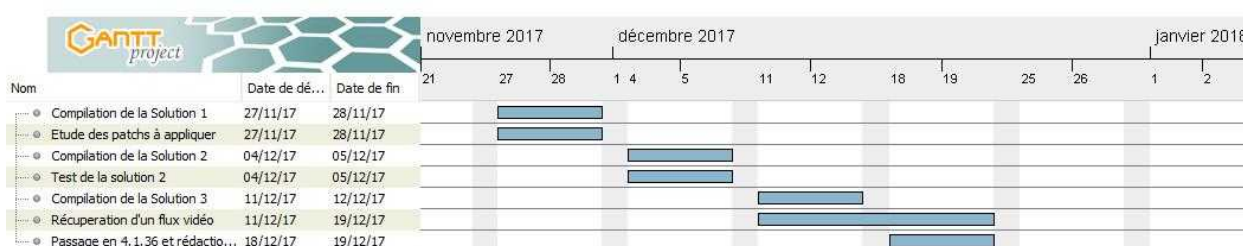


FIGURE 1.1 – Avancement du projet actuel

Les trois solutions qui sont présentées ci-dessous correspondent à trois codes sources.

Chapitre 2

Solution n° 1

Plateforme : i.MX6

Version du Kernel : 3.14

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

Pour décorréliser les erreurs de compilation des erreurs de manipulation Yocto, nous compilons les fichiers depuis la cross-toolchain du SDK généré par Yocto. Décrit dans \$CC.

```

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

```

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

```

Inclusion de 6 dossiers de bibliothèques en option puis, nouvelles dépendances inexistantes dans le dossier contenant l'arborescence de compilation (builddir). Recherche d'une autre solution grâce au dialogue avec le professeur de Linux embarqué. Conseil retenu : Compilation dans l'arborescence Yocto (in-tree) directe. Certains fichiers restent inexistantes dans le kernel 3.14 donc, patch depuis le kernel 4.1.38.

Chapitre 3

Solution n° 2

Plateforme : i.MX6

Version du Kernel : 3.14

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

3.1 Fichiers sources

Ce driver contient les fonctions de configuration de l'imx219 : on, off, gain, couleurs, taille de l'affichage. Il repose sur les couches de driver i2c et v4l. La configuration du périphérique imx219 effectuée par v4l-utils est automatique lors du chargement du module. Elle s'appuie sur les trois couches de driver imx219>v4l>i2c. Le flux vidéo configuré peut ensuite être récupéré par Gstreamer.

3.2 Travail effectué

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.

Création d'une recette

3.3 L'arborescence des fichiers in-tree

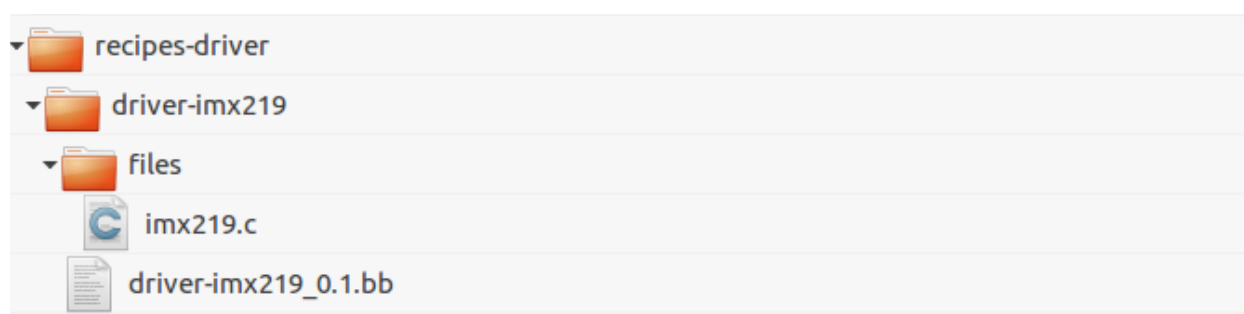


FIGURE 3.1 – Arborescne des fichiers

3.4 Compilation de driver

Extrait de la recette : récupère le fichier `imx219.c`, le compile par la cross tool-chain et l'installe

```

1 SUMMARY = "driver_imx219"
2 SECTION = "examples"
3 LICENSE = "MIT"
4 LIC_FILES_CHKSUM =
5     "file://${COMMON_LICENSE_DIR}/MIT;md5=c2b5f7071fdde268bf46ace1546f3c4b"
6 SRC_URI = "file://imx219.c"
7
8 S = "${WORKDIR}"
9
10 do_compile() {
11     ${CC} imx219.c -o imx219
12 }
13
14 do_install() {
15     install -d ${D}${bindir}
16     install -m 0755 imx219 ${D}${bindir}
17 }

```

3.5 Utilisation du driver sur la cible

Vérification de la présence du driver dans l'arborescence :

```

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

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

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! autovi-  
deosink
```

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

Actuellement nous cherchons l'utilisation des commandes `gst-launch-1.0` et des fichiers de configurations `v4l2src`. Poursuite de cette piste dans le chapitre 5.

Chapitre 4

Solution n° 3

Plateforme : Hummingboard (i.MX6)

Version du Kernel : 4.13

Source : <http://git.armlinux.org.uk/cgit/linux-arm.git/commit/?h=csi-v6&id=e3f847cd37b007d55b7628241>
<http://git.armlinux.org.uk/cgit/linux-arm.git/commit/?h=csi-v6&id=4bd8e1231a2e6eca6a65b565176e>

Les sources étant sous forme de patch nous allons cette fois directement les intégrer à notre recette.

4.1 Contenu des sources

Kconfig : ajoute l'imx219 au menuconfig, pour demander la compilation du driver. Utilisation :

1. Modifier le menuconfig
2. Récupérer le defconfig
3. Ajouter à notre recette.

Makefile : ajoute la compilation du driver (pourra être activée par le defconfig)

Imx219.c : driver qui contient l'ensemble des configurations spécifiques à la caméra.

Device-tree pour Hummingboard : annonce les périphériques avec lesquels le driver aura des interactions (i2c, csi, gpio, clk).

4.2 Problèmes rencontrés

Des libraires 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 patchs sur une version plus récente soit 4.1.36 (Krogoth sous Yocto).

4.3 lien GIT

Vous trouverez ici un lien qui contient l'ensemble des patchs appliqués sur la version 3.14 du kernel : <https://github.com/Alanaitali/meta-openrexplicam/tree/develop/gladwistor/recipes-kernel/linux>

4.4 Source des patches

racine de l'adresse : [http://git.armlinux.org.uk/cgit/linux-arm.git/commit/?h=csi-v6
&id=e3f847cd37b007d55b76282414bfcf13abb8fc9a
&id=4bd8e1231a2e6eca6a65b565176ea9722611c8dd](http://git.armlinux.org.uk/cgit/linux-arm.git/commit/?h=csi-v6&id=e3f847cd37b007d55b76282414bfcf13abb8fc9a&id=4bd8e1231a2e6eca6a65b565176ea9722611c8dd)

4.5 Ajout de paquets

En prévision d'un usage ultérieur de v4l-utils et/ou gstreamer comme applicatif (user-space), nous avons essayé de les compiler. Cette compilation n'a abouti ni par compilation in-tree ni out-of-tree, ni par une recette Yocto. Par la suite nous avons ajouté ces paquets entre autres 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     gstreamer \
8     i2c-tools \
9     gstreamer1.0-plugins-imx \
10    gst1.0-fsl-plugin \
11    v4l-utils \
12    "
```

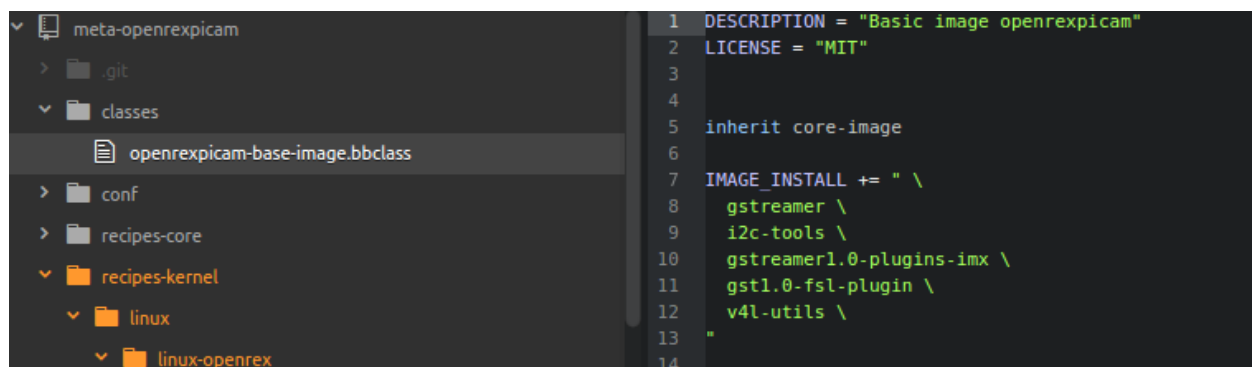


FIGURE 4.1 – Emplacement et contenu de la recette de l'openrexpica-base-image

Chapitre 5

État de l'art de la récupération du flux vidéo

5.1 Methodologie par la Webcam USB

Nous nous agüerissons à l'utilisation de Video4Linux-utils, de plus nous avons trouvé un journal en ligne expliquant la récupération du flux vidéo d'une webcam USB avec V4L2-utils sur imx6. Dans un premier temps, nous avons capturé le flux d'une webcam USB intégrée à un de nos ordinateurs personnels afin de comprendre les commandes et paramètres. Ensuite nous avons récupéré une webcam USB lambda afin de tester les commandes sur la carte openrex et essayer de capturer le retour vidéo. Une fois fonctionnel nous passerons sur le flux vidéo de la Raspi Cam v2.

1. Récupérer le port qui est connecté la webcam USB grâce à 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=uvccvideo, 480M
      |__ Port 2: Dev 3, If 1, Class=Video, Driver=uvccvideo, 480M
```

FIGURE 5.1 – Port USB de la webcam USB

2. La webcam USB est sur le bus I2C n°1 avec comme identifiant de 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

3. Il est nécessaire d'écrire le numéro du bus et l'identifiant du port dans les fichiers bind et unbind. Le fichier bind permet 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
uvccvideo: Found UVC 1.00 device Microsoft LifeCam (045e:074a)
input: Microsoft LifeCam as /devices/soc0/soc.0/21000000.aips-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

4. La webcam USB est bien reconnu par notre système qui affiche sa référence (cf. Figure 5.3).

5. Démarrer un flux vidéo sur notre webcam USB s'effectue avec la commande : **# gst-launch-1.0 v4l2src device="/dev/video0" ! video/x-raw,w idth=640,height=480 ! autovideosink**

```
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

6. Nous sommes persuadés de la bonne capture du flux vidéo puisque lorsque nous la lançons, le retour de la commande se fige à :

```
1 New clock GstSystemClock
```

C'est exactement le même endroit quand nous lançons la même commande sur notre propre ordinateur avec notre webcam intégrée. De plus une petite LED verte s'allume au moment de l'envoi de cette commande (cf. Figure 5.5).



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 en protocole UDP.

5.3 Point d'amélioration

Pour commencer il est nécessaire de mettre l'image de l'openrexp à jour afin de se connecter à un réseau WiFi ensuite 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 se pilote par le CSI et non l'USB.

5.4 Sources

Lien du blog de la récupération du flux vidéo par la webcam USB :

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

Chapitre 6

Conclusion

Solution 1 : la compilation out-of-tree est abandonnée, les sources sont en revanche toujours susceptibles d'être utilisées.

Solution 2 : Cette solution est notre piste la plus concrète. En effet la compilation a réussi, nous cherchons à comprendre comment utiliser le driver à travers v4l. Une fois cette tâche effectuée nous travaillerons sur le portage des versions du kernel (3.14 → 4.1)

Solution 3 : La compilation du driver n'a pas fonctionné car les bibliothèques kernel et celles requises par le driver ne sont pas compatibles. Une autre solution serait d'essayer la même compilation sur un kernel 4.1.

De part l'avancée que nous avons actuellement et l'expérience accumulée au cours du projet il nous reste plusieurs pistes à explorer. Sans contre-indication particulière nous devons être capables d'avoir des résultats dans quelques semaines. 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.