

Système embarqué pour véhicule intelligent

Projet PRIM (PIM380 2015/2016)

Membres: Artur A. M. Sarlo, Chenxi Du, Jie Chen, Yushan Sun

[1. Introduction](#)

[2. Evaluation de design](#)

[2.1 Design matériel](#)

[2.2 Design logiciel](#)

[3. Choix de plate-forme](#)

[3.1 Carte ZYBO](#)

[3.2 Choix du support](#)

[3.3 Ubuntu sur ZYBO](#)

[4. Architecture](#)

[4.1 Chaîne de traitement d'image pour ARM](#)

[4.2 Système de contrôle des périphériques](#)

[5. Linux driver](#)

[5.1 Driver de VDMA](#)

[5.2 Driver de PS2PL et PL2PS](#)

[6. Traitement d'image](#)

[6.1. Introduction des outils](#)

[6.2 Traitement en logiciel](#)

[6.3 Traitement en matériel](#)

[6.4 Résultats](#)

[7 Application portable](#)

[7.1. Architecture](#)

[7.2 Implémentation](#)

[7.3 Challenge](#)

[7.4 Résultat et Conclusion](#)

[7.5 A faire plus loin](#)

[8. Conclusion](#)

1. Introduction

Aujourd’hui , les moyens de transport intelligents comme les ‘Smart Cars’ ne sont plus des choses impensables. Par exemple , les ‘Google Cars’ ont déjà été présentés au public. De plus, Apple a présenté le développement d’une voiture intelligente dans son plan d’entreprise pour les années à venir. Les produits intelligents sont de plus en plus populaires dans le marché de l’électronique pour le grand public, y compris dans les moyens de transport utilisés au quotidien. Afin de comprendre et de maîtriser les technologies mises en oeuvre dans ce domaine, nous avons décidé d’entreprendre un projet autour d’un prototype de voiture intelligente. Ce véhicule autonome pourra identifier des obstacles dans son champ de vision , choisir le chemin optimal et aller à des endroits prédéterminés en envoyant le vidéo vers portable .

2. Evaluation de design

Dans notre système a deux parties: une partie matérielle et une partie logicielle. Afin d’assurer un fonctionnement correct du système, ces deux parties doivent être bien coordonnées.

2.1 Design matériel

Normalement, dans un système de voiture, il existe deux parties principales: une partie pour contrôler le mouvement de voiture, par exemple, moteur et freinage etc, et l’autre partie pour prendre décision, par exemple, il doit tourner à gauche selon le traitement d’image. Afin d’assurer un fonctionnement correct du système, ces deux parties doivent être bien coordonnées(figure 1).

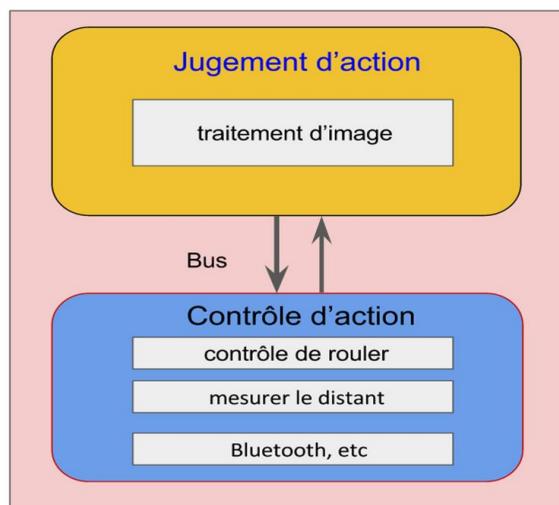


Figure 1. Evaluation d’architecture principale du système

Les deux parties sont indépendantes mais doivent pouvoir communiquer tout en garantissant que les fonctions critiques fonctionnent correctement.

Dans ce projet on a deux grandes contraintes principales:

I) Il y aura un ensemble de tâches que s’exécutera en permanence:

- la voiture doit pouvoir rouler en continu,
- le contrôle des moteurs de chaque roue doit être fait en permanence,

- la mesure de la distance parcourue doit être faite en continu,
- la communication avec un téléphone portable par bluetooth etc,

Tableau 1. Responsabilités des deux parties séparées

Partie de contrôle des actions	Partie de prendre décision
contrôle des moteurs	choix de l'itinéraire
détection des situations d'urgence (blocage, chute...)	analyse de l'environnement
contrôle de la direction	...

II) En même temps, il faut analyser l'environnement et choisir le bon chemin pour éviter les obstacles. Cette analyse se fera en traitant les images provenant d'une caméra. On sait que la tâche de traitement d'image est beaucoup plus lourde que les tâches de contrôle du mouvement.

Les deux ensembles de tâches critiques seront faites de façon indépendante sur deux processeurs différents, avec des contraintes de puissance de calcul et de temps réel différentes:

- un processeur (puissant) applicatif pour le traitement d'image et l'interaction avec l'utilisateur
- un microcontôleur (MCU) pour la gestion temps-réel de la voiture

2.1.1 La partie de jugement d'action

Puisque la partie de prendre décision(jugement) est la plus importante dans notre système, il y a un cœur fort qui peut faire le calcul très rapidement. Selon la figure 2, au tour du cœur MCU, il y a des composants qui consistent un petit système. La caméra est utilisé à obtenir l'image originale. Les flash sont considérés car on va stocker des buffers d'image envoyé par la caméra. Le 'sram' sera ajouté si le programme est gros.

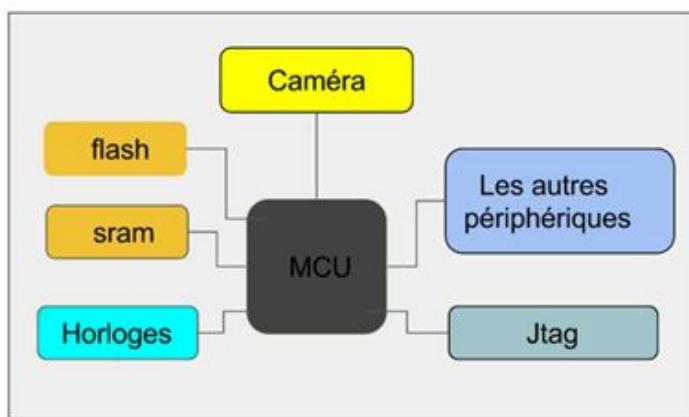


Figure 2. La partie de jugement

Pour programmer/debugger sur la carte, on va utiliser le JTAG pour faire communiquer l'ordinateur et MCU. En fin, s'il y a d'autres fonctions, d'autres périphériques peuvent être considérés.

2.1.2 Partie de contrôle d'action

Dans cette partie là, puisque il y a moins de travail à faire, on va utiliser un MCU de performance normale. Cette coeur fait des tâches légères. Autour de celui là: une interface de contrôle pour les moteurs de chaque roue et une autre pour mesurer la distance parcourut par la voiture (à partir du nombre de tours des roues).

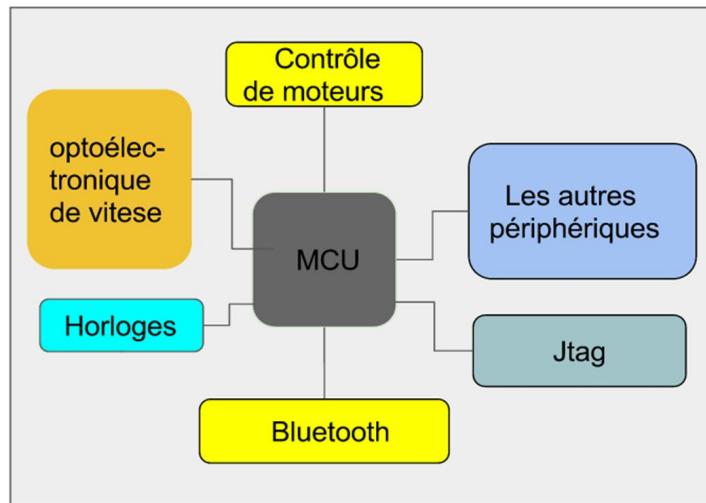


Figure 3. La partie de contrôle

De plus, l'interface de Bluetooth permet la communication entre le portable et ce système. Plusieurs interactions peuvent être explorées une fois que l'environnement embarqué est mis en ouvre. L'interaction avec une application mobile est probablement le plus évident.

2.1.3 Communication entre les deux parties

Le bus entre les deux parties permet la communication entre les deux. Mais, le choix de protocole de bus n'est pas déterminé. Les deux parties fonctionnent ont différentes fréquences (la fréquence de la partie de jugement est plus vite que celui de la partie de contrôle).

2.2 Design logiciel

Pour bien coopérer avec la partie matérielle, le logiciel doit contenir quatre fonctionnalités.

2.2.1 GUI et application de Androïde

Actuellement les applications pour les smart phone sont très à la mode, donc développer une application pour contrôler et gérer la voiture semble pratique et très accessible. Dans notre projet, on va développer une application en Androïde. Elle pourra contrôler la voiture, bien sur, mais en exploitant les fonctionnalités d'interactions disponibles pour les smartphones on espère ajouter des nouvelles fonctionnalités.

2.2.2 Système de contrôle

La première fonction est le système de contrôle. Comme fonctionnalité fondamentale, on peut contrôler la voiture: la faire rouler, tourner et s'arrêter. Il faut qu'on puisse bien contrôler la voiture et aussi savoir son état. On doit faire l'architecture de logiciel en utilisant l'UML, et

aussi le vérifier et valider.

2.2.3 Système de suivi

C'est un objectif avancé, une fois quelle peut rouler, on a l'ambition qu'elle puisse rouler elle même sans intervention humaine! En conséquence, on veut développer un système de suivi, c'est à dire, en se déplaçant, elle pourra nous suivre sans aucune instruction directe. C'est un challenge, on doit développer un algorithme que puisse détecter le maître, on sera dépendant aussi de la partie détection d'obstacles, que devra être bien implémentée pour permettre l'existence de cette fonctionnalité.

2.2.4 Interface Bluetooth

Après développer le système logiciel, on doit faire les connexions avec la voiture. En réalisant cela, on utilisera la technologie Bluetooth. On échangera des petits messages, précisément, des signaux entre l'application et la voiture. Pour bien contrôler la voiture, elle devra être capable d'interpréter les signaux de contrôle et d'émettre son état et ses données de roulement à l'application. Comment gérer et analyser ces données est aussi un sujet pour nous.

2.2.5 Interface vidéo

L'utilisateur s'intéressent bien à l'affichage de vidéo pour voir ce que la voiture voit. Il est très agréable d'ajouter ce fonctionnalité.

3. Choix de plate-forme

Nous savons tous que la réalisation de traitement de l'image dans une plate-forme portable est toujours difficile en raison de la limitation de la performance, en temps réel, de surface, la consommation d'énergie, etc. Avec le microprocesseur normal, il est donc difficile de réaliser le traitement en temps réel, même avec l'image de mauvaises qualités, sans parler de la HD, VGA. Aussi, si nous voulons utiliser un algorithme complexe de traitement d'image, il est presque impossible. Donc, avec l'arrière-plan et la conception de systèmes sur puce, nous pouvons bénéficier de deux avantages de matériel et de logiciels de réaliser un système de coopération de traitement d'image.

Par conséquent, nous avons besoin de choisir une plate-forme qui contient à la fois le FPGA et le système de programme qui est flexible à la conception. Ainsi, la famille de Zynq est un bon choisissent de compléter notre projet. Ainsi, la famille de Zynq est un bon choisissent de compléter notre projet. De les caractéristiques de la Zynq-7010, il répond à notre demande.

Tableau 2. Fonctionnalités de Zynq 7010 AP SoC

650Mhz dual-core Cortex-A9 processor	Reprogrammable logic equivalent to Artix-7 FPGA
DDR3 memory controller with 8 DMA channels	4,400 logic slices , each with four 6-input LUTs and 8 flip-flops
High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO	240 KB of fast block RAM
Low-bandwidth peripheral controller: SPI, UART, CAN, I2C	Two clock management tiles, each with a phase-locked loop (PLL) and mixed-mode clock manager (MMCM) 80 DSP slices

	Internal clock speeds exceeding 450MHz
	On-chip analog-to-digital converter (XADC)

3.1 Carte ZYBO

Le ZYBO (Zynq Board) est un entrée de gamme logiciel riche en fonctionnalités prêtes à l'emploi intégré et plate-forme de développement de circuit numérique construit autour du plus petit membre de la famille Xilinx Zynq-7000, le Z-7010. Le Z-7010 est basé sur l'architecture du Xilinx System-on-Chip (AP SoC), qui intègre un processeur ARM étroitement Cortex-A9 dual-core avec Xilinx série 7 Field Programmable Gate Array (FPGA) logique. Lorsque couplé avec le riche ensemble de multimédia et la connectivité des périphériques disponibles sur le ZYBO, l'Zynq Z-7010 peut accueillir une conception de l'ensemble du système. La on-board mémoire, vidéo et audio I / O, double rôle USB, Ethernet, et slot SD auront votre conception-et-prêt sans matériel supplémentaire nécessaire. En outre, six connecteurs Pmod sont disponibles pour mettre toute la conception de designer facilement^[1].

Tableau 3. Fonctionnalités de ZYBO

ZYNQ XC7Z010-1CLG400C
512MB x32 DDR3 w/ 1050Mbps bandwidth
Dual-role (Source/Sink) HDMI port 16-bits per pixel VGA source port
Trimode (1Gbit/100Mbit/10Mbit) Ethernet PHY
MicroSD slot (supports Linux file system)
OTG USB 2.0 PHY (supports host and device) External EEPROM (programmed with 48-bit globally unique EUI-48/64™ compatible identifier) Audio codec with headphone out, microphone and line in jacks 128Mb Serial Flash w/ QSPI interface On-board JTAG programming and UART to USB converter
GPIO: 6 pushbuttons, 4 slide switches, 5 LEDs
Six Pmod connectors (1 processor-dedicated, 1 dual analog/digital, 3 high-speed differential, 1 logicdedicated)

De l'introduction, nous pouvons voir que la carte a le soutien de l'Ethernet, mémoire 512M DDR3, VGA et HDMI soutien, et 6 Interface Pmod qui est très important que il nous permet d'élargir et ajoutons nos périphériques flexible.

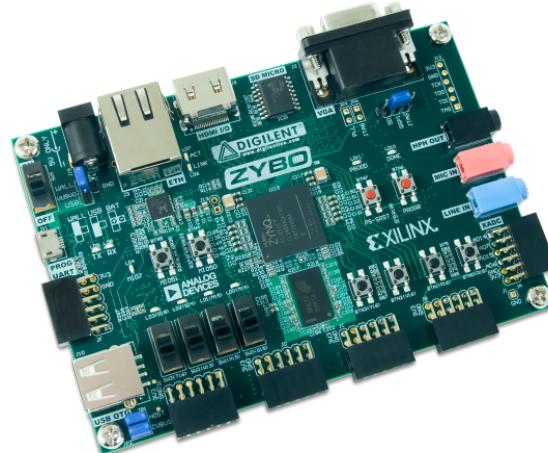


Figure 4. Carte ZYBO avec Zynq-7000

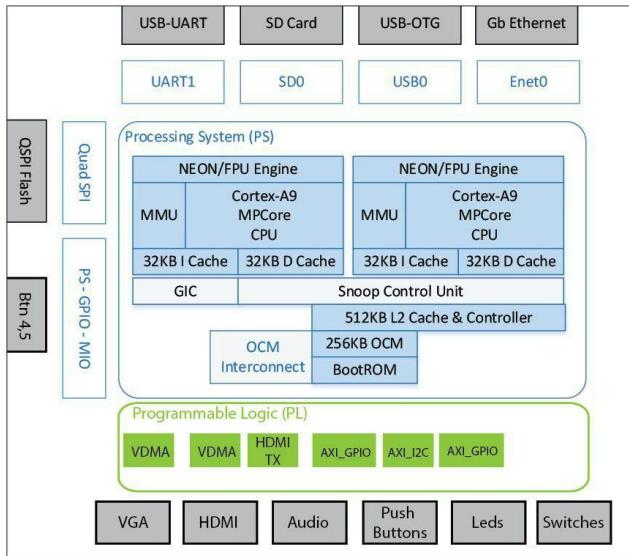


Figure 5. Référence de base architecture du système matériel pour ZYBO

3.2 Choix du support

Il est une question que nous allons utiliser le système en métal nu, sans beaucoup de soutien ou d'utiliser un système avec beaucoup de ressources. Il est clair que d'un système en métal nu, il est plus facile de faire un peu d'optimisation, mais il est difficile pour le développement. si nous utilisons le système comme Linux embarqué, il est plus souple et plus facile à configurer d'Ethernet ou Wifi. De plus, nous pouvons également utiliser la bibliothèque OpenCV de traitement de l'image. Par conséquent, nous avons choisi le système Linux avec le support d'Ubuntu. Les étapes pour réaliser le système Ubuntu dans ZYBO sont au-dessous.

3.3 Ubuntu sur ZYBO

3.3.1 Flot de Xilinx design

La figure montre un schéma du flux de conception Xilinx pour Zynq AP SoC de bloc de haut niveau. Nous avons ici trois parties principales, la première partie est "system hardware design" avec Vivado. Nous avons besoin de configurer le PS(processing system) qui contient un processeur d'ARM Cortex-A9 dual-core et de créer ou d'ajouter certains IPs nécessaires pour remplir notre objectif. En fin de compte, après la synthèse et de routage, nous pouvons obtenir le bitstream cible qui peut être utilisé pour générer FSBL et 'devicetree' en utilisant Xilinx SDK. Pour la deuxième partie, nous avons besoin d'aller chercher des sources pour u-boot, le noyau linux ou disque virtuel si nous ne pas utiliser le système de fichiers racine de ubuntu et les compiler avec des configurations correctes pour obtenir 'ulmage', 'u-boot', ou 'uRamdisk' qui peut être utilisé pour créer 'BOOT.bin' avec le FSBL et le bitstream créés par première partie.

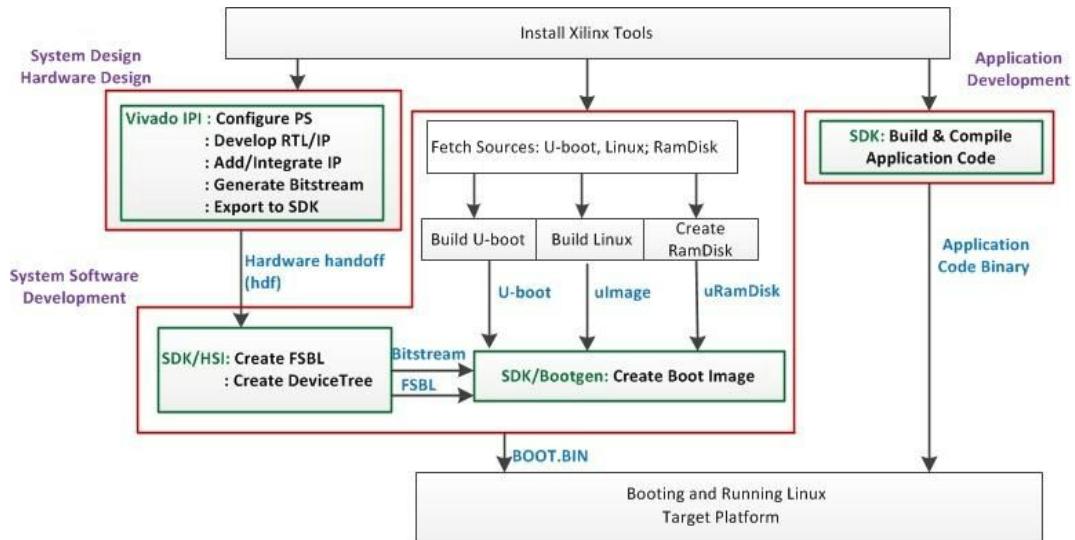


Figure 6. High level block diagram of the Xilinx design flow for Zynq AP SoC

A la fin, nous avons besoin de programmer pour démarrer et contrôler le matériel pour réaliser notre objectif. Pour notre projet, nous avons besoin d'écrire le code c pour configurer le matériel et d'utiliser OpenCV pour le traitement de l'image.

3.3.2 Etapes

Prérequis:

Outil : Vivado 2015.2, Ubuntu 14.04, SD card de 4G,

Pour compiler l'U-boot et plus tard le noyau, nous devons mettre en place les outils et prolonger la variable d'environnement '\$PATH' pour trouver les outils. Nous recommandons de déplacer les deux premières exportations et la source dans votre '.bashrc' de sorte que vous ne devez pas les régler à nouveau après la fermeture du terminal.

```

File Edit View Search Terminal Tabs Help
owen@ubuntu:~ x owen@ubuntu:~ x
owen@ubuntu:~$ export ARCH=arm
owen@ubuntu:~$ export CROSS_COMPILE=arm-xilinx-linux-gnueabi-
owen@ubuntu:~$ export PATH=$PATH:/home/owen/linux/zybo/u-boot-Digilent-Dev/tools
owen@ubuntu:~$ source ~/petalinux/petalinux-v2015.2.1-final/settings.sh
  
```

The screenshot shows a terminal window on an Ubuntu system. The user has run several commands to set environment variables. The first command is 'export ARCH=arm', followed by 'export CROSS_COMPILE=arm-xilinx-linux-gnueabi-'. Then, the user sets the PATH variable to include a directory containing tools. Finally, the user runs 'source ~/petalinux/petalinux-v2015.2.1-final/settings.sh' to apply these changes. The terminal window has a dark background and light-colored text. The title bar shows the user's name and the terminal session.

Figure 7. Variables d'environnement

3.3.2.1 U-boot compilation^[2] [3]

Conditions préalables :

Cross compilateur : arm-xilinx-linux-gnueabi-

(1) Obtenez le code source pour les U-Boot à partir du référentiel Digilent Git :

`git clone https://github.com/DigilentInc/u-boot-Digilent-Dev.git`

(2) L'étape suivante consiste à configurer et compiler le U-Boot. Accédez au dossier u-boot-Digilent-Dev dans le terminal. avant de compiler l'u-boot, nous devons changer le fichier *zynq_zybo.h* pour empêcher l'u-boot de charger le disque virtuel. Au lieu de cela, nous voulons le faire passer au système de fichiers racine^[3]. si vous ne trouvez pas le 'zynq_zybo_config', trouvez s'il existe le fichier 'zynq_zybo.h' dans le répertoire '*u-boot-Digilent-Dev/include/configs/*', sinon copiez ce fichier https://github.com/DigilentInc/u-boot-Digilent-Dev/blob/master-next/include/configs/zynq_zybo.h et l'y mettre.

```

82      "sdboot;if mmcinfo; then \" \
83          \"run uenvboot; \" \
84          \"echo Copying Linux from SD to RAM... && \" \
85          \"fatload mmc 0 0x3000000 ${kernel_image} && \" \
86          \"fatload mmc 0 0x2A00000 ${devicetree_image} && \" \
87          \"bootm 0x3000000 - 0x2A00000; \" \
88          \"fi\" \

```

Figure 8. Configuration U-boot

(3) Pour compiler U-Boot, nous avons besoin d'outils de compilation croisée qui sont fournis par Vivado 2015.2. Ces outils ont un préfixe bras *xilinx-linux-gnueabi-* aux noms standard pour la chaîne d'outils de GCC. Le préfixe référence les plates-formes qui sont utilisées. Le conseil ZYBO a deux noyaux de bras, de sorte que nous référencer bras. Pour utiliser les compilateurs multi-plateformes, s'il vous plaît assurez-vous que les paramètres Vivado 2015.2 ont été source. Si non, nous vous recommandons de télécharger ce compilateur sur [http://www.xilinx.com/guest_resources/member\(mb_gnu/xilinx-2014.11-30-arm-xilinx-linu x-gnueabi.src.tar.bz2](http://www.xilinx.com/guest_resources/member(mb_gnu/xilinx-2014.11-30-arm-xilinx-linu x-gnueabi.src.tar.bz2) et installer selon <http://www.wiki.xilinx.com/Install+Xilinx+Tools> Pour configurer et construire des U-Boot pour ZYBO, suivez le command : **make CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_zybo_config**

(4) Après :

make CROSS_COMPILE=arm-xilinx-linux-gnueabi-

Nous vous suggérons d'ajouter les deux phrase dans votre paramètre environnement.

export ARCH=arm

export CROSS_COMPILE=arm-xilinx-linux-gnueabi-

(5) Après la compilation, l'ELF (Executable and Fichier Linkable) généré est nommé u-boot. Nous devons ajouter une extension .elf le nom de fichier afin que Xilinx SDK peut lire le format de fichier et de générer BOOT.BIN.

3.3.2.2 Génération de BOOT.bin^[2]

- (1) Export de la conception de matériel (après l'article 1, l'étape 16) pour Xilinx SDK dans Vivado en cliquant sur 'File -> Export -> Export Hardware for SDK -> include bitstream'. Cliquez 'Launch SDK' pour ouvrir Xilinx SDK.
- (2) Après SDK lance, le projet de plate-forme matérielle est déjà présent dans l'Explorateur de projet sur la gauche de la fenêtre principale SDK, comme le montre la figure. 38. Nous avons besoin maintenant de créer une première étape Bootloader (FSBL). Cliquez sur File-> New-> Project ...,
- (3) Dans la fenêtre **New Project**, sélectionnez Xilinx-> Application project, puis cliquez sur **Next.**

(4) Nous allons nommer le projet FSBL(First Stage Boot Loader). **Next**.

(5) Sélectionnez **Zynq FSBL** comme modèle, et cliquez sur **Finish**.

(6) Nous avons le fichier fsbl.elf et votre_nomxxx.bit et u-boot.elf, nous pouvons créer le fichier BOOT.bin en utilisant l'outil 'create Zynq Boot image' dans xilinx SDK .

3.3.2.3 Compilation du noyau Linux^[2]

(1) Récupérer le code source du noyau Linux à partir Digilent dépôt Git.

git clone <https://github.com/DigilentInc/Linux-Digilent-Dev.git>

(2) Nous allons commencer à configurer le noyau avec la configuration par défaut pour ZYBO. La configuration est situé au *arch/arm/configs/xilinx_zynq_defconfig*. Pour utiliser la configuration par défaut,

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- xilinx_zynq_defconfig

Si vous voulez configurer votre propre noyau, juste essayer

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- menuconfig

Après,

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-

(3) Après la compilation, l'image du noyau est situé au *arch/arm/boot/zImage*. Cependant, dans ce cas, l'image du noyau doit être un ulimage (décompressé) plutôt que d'un ulimage. Pour rendre le ulimage,

make ARCH=arm UIMAGE_LOADADDR=0x8000 ulimage

3.3.2.4 Génération du Device Tree Blob^[3]

(1) Avant que nous générions l'arbre blob de l'appareil, nous devons faire quelques ajustements au fichier zynq-zybo.dts trouvé *Linux-Digilent_dev/arch/arm/boot/dts*. Particulièrement la ligne 44, 53, et 62. changer l'horloge empêché une erreur qui est survenue après le démarrage jusqu'à root shell.

```
41 chosen {
42     /*bootargs = "console=ttyPS0,115200 root=/dev/ram rw earlyprintk";
43     bootargs = "console=ttyPS0,115200 root=/dev/ram rw initrd=0x8000000,8M init=/init earlyprintk rootwait devtmpfs.mount=1";*/
44     bootargs = "console=ttyPS0,115200 root=/dev/mcblk0p2 rw earlyprintk rootfstype=ext4 rootwait";
45     linux,stdout-path = "/amba@0/serial@e8001000";
46 };
47 cpus {
48     #address-cells = <1>;
49     #size-cells = <0>;
50     ps7_cortexa9_0: cpu@0 {
51         bus-handle = <&ps7_axi_interconnect_0>;
52         clock-latency = <1000>;
53         clocks = <&clkc_2>;
54         compatible = "arm,cortex-a9";
55         device_type = "cpu";
56         interrupt-handle = <&ps7_scugic_0>;
57         operating-points = <666667 1000000 333334 1000000 222223 1000000>;
58         reg = <0x0>;
59     };
60     ps7_cortexa9_1: cpu@1 {
61         bus-handle = <&ps7_axi_interconnect_0>;
62         clocks = <&clkc_2>;
63         compatible = "arm,cortex-a9";
64         device_type = "cpu";
65         interrupt-handle = <&ps7_scugic_0>;
66         reg = <0x1>;
67     };
68 }
```

Figure 9. Choix de démarrage

```

owen@ubuntu:~/linux/zybo/Linux-Digilent-Dev
owen@ubuntu:~/linux/zybo/Linux-Digilent-Dev$ ./scripts/dtc/dtc -I dts -O dtb -o ../devicetree.dtb arch/arm/boot/dts/zynq-zybo.dts
owen@ubuntu:~/linux/zybo/Linux-Digilent-Dev$ 

```

Figure 10. Compilation de DeviceTree

3.3.2.5 Partitionnement de votre SD carte

Il existe deux façons de le faire, le plus simple est d'utiliser Gparted. Une deuxième façon consiste à utiliser l'outil fdisk de ligne de commande^[4]. La première partition doit être FAT32 et être 1 Gio, la seconde partition ext4 devrait être et peut prendre l'espace restant.

```

owen@ubuntu: ~
owen@ubuntu:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0   20G  0 disk
└─sda1   8:1    0   16G  0 part /
  └─sda2   8:2    0     1K  0 part
  └─sda5   8:5    0     4G  0 part [SWAP]
sdb      8:16   1   3.7G  0 disk
└─sdb1   8:17   1     1G  0 part /media/owen/ZYBO_BOOT
  └─sdb2   8:18   1   2.7G  0 part /media/owen/ROOT_FS
sr0     11:0    1  1024M 0 rom
sr1     11:1    1  1024M 0 rom
owen@ubuntu:~$ 

```

Figure 11. Partition de SD carte

3.3.2.6 Renouvellement de Ubuntu^[5]

Maintenant, nous devons mettre BOOT.bin, devicetree.dtb et ulimage sur la partition de démarrage FAT32. Nous recommandons la création d'un dossier temporaire, le montage de la partition de démarrage à ce dossier, puis en utilisant rsync pour copier les fichiers. Une fois les fichiers sont sur la bonne partition éjecter la carte SD et la mettre dans votre ZYBO. Assurez-vous que le cavalier JP5 est réglé sur SD et JP7 est fixé au mur. Pour se connecter à un terminal de port UART, je l'utilise *Tera Term* se connecter au matériel. Allumez le ZYBO puis dans le terminal du port série définir la configuration du port où l'ZYBO est connecté et a fixé le taux de transmission à 11500, tout le reste doit rester par défaut. Le rootfs se trouve sur : <http://www.wiki.xilinx.com/Ubuntu+on+Zynq>

```

File Edit Setup Control Window Help
Ubuntu 12.10 zynq ttyPS0
zynq login: root
Password:
Last login: Wed Dec 31 16:04:21 PST 1969 from 192.168.1.114 on pts/1
Welcome to Ubuntu 12.10 (GNU/Linux 3.18.0-xilinx-46106-g3913e72 armv7l)

 * Documentation: https://help.ubuntu.com/
root@zynq: # dmesg
Booting Linux on physical CPU 0x0
Linux version 3.18.0-xilinx-46106-g3913e72 (owen@ubuntu) (gcc version 4.9.1 (Sourcery CodeBench Lite 2014.11-30) ) #4 SMP PREEMPT Fri Jan 15 17:18:36 AST 2016
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: Xilinx Zynq
crn: Reserved 16 MiB at 0x1a800000
Memory policy: Data cache writealloc
On node 0 totalpages: 114688
free_area_init_node: node 0, pgdat 4069bd0, node_mem_map 5bc70000
Normal zone: 896 pages used for memmap
Normal zone: 0 pages reserved
Normal zone: 114688 pages, LIFO batch:31
PERCPU: Embedded 10 pages/cpu 0xbcb53000-0xbcb53760
pcpu-alloc: 0xbcb53000-0xbcb53760 pcpu-alloc: 0xbcb53760-0xbcb53800 alloc=10*4096
pcpu-alloc: [0] 0 [0] 1
Built 1 zonelists in zone order, mobility grouping on. Total pages: 113792
Kernel command line: console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk mem=448M rootfstype=ext4 rootwait devtmpfs.mount=0
P10 hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode cache hash table entries: 32768 (order: 5, 131072 bytes)

```

Figure 12. Information de Linux boot

Après cela, nous avons besoin d'installer certains logiciels essentiels, toolchain, et une bibliothèque, etc. Cependant, parce que la version Ubuntu 12.10 est hors de la mise à jour, nous avons besoin de mettre à jour les dépôts Ubuntu (de source.list) avec les référentiels suivants:

```
"deb http://old-releases.ubuntu.com/ubuntu/ quantal main restricted universe multiverse
deb http://old-releases.ubuntu.com/ubuntu/ quantal-security main restricted universe
multiverse
deb http://old-releases.ubuntu.com/ubuntu/ quantal-updates main restricted universe
multiverse"
$ vi /etc/apt/sources.list // [add repositories]
$ apt-get upgrade
$ apt-get update
```

Ensuite, nous pouvons télécharger ce que nous avons besoin. Par exemple :

```
$ apt-get install build-essential
$ apt-get install libssl-dev
$ apt-get install u-boot-tools
$ apt-get install libopencv-dev
```

4. Architecture

Selon la section 2, nous savons qu'il y a deux parties dans notre conception. Maintenant, avec le support de SoC ZYNQ, la partie de prendre la décision utilise processeur ARM dans ZYBO, et l'autre partie de contrôleur de mouvement utilise un soft cœur MicroBlaze grâce au FPGA dans ZYNQ.

La figure 13 nous montre qu'il y a trois parties dans le FPGA, le processeur MicroBlaze, le chaîne de traitement d'image qui prépare l'image nécessaire pour logiciel, et deux composantes qui permettent la communication entre l'ARM et MicroBlaze. Les deux modules qui s'appellent 'PL2PS' et 'PS2PL' sont créés par nous en utilisant le protocole AXI4 et ils utilisent interruption pour échanger l'information. Cet interruption est ajouté dans Linux que nous allons la détailler dans la section suivante.

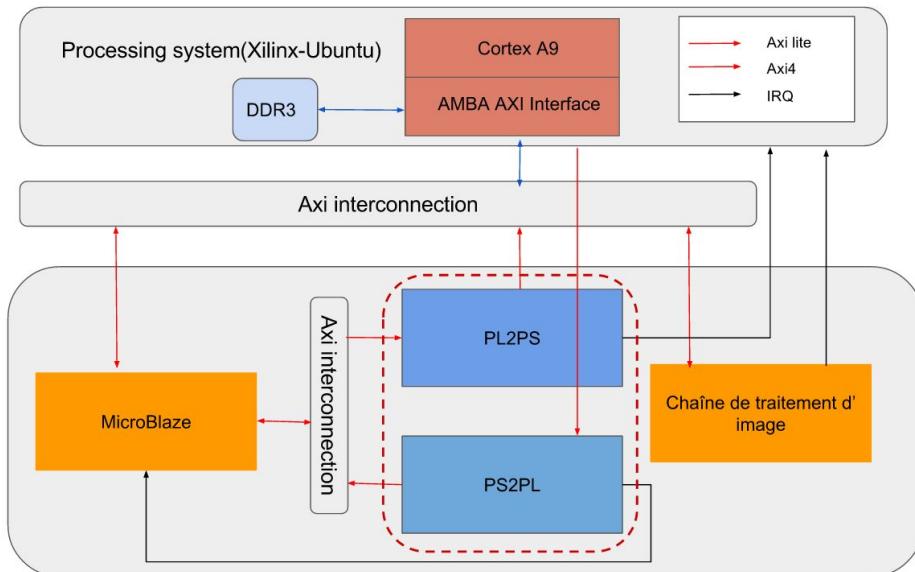


Figure 13. Architecture global

4.1 Chaîne de traitement d'image pour ARM

Dans cette partie, nous allons vous présenter le design dans la partie logique programmable. Nous allons utiliser le bitstream pour configurer le FPGA pour réaliser notre objectif. Par exemple, nous avons besoin d'écrire un pilote pour le OV7725 de la caméra, et de transformer le flux VGA à débit AXI-Stream standard, et ensuite utiliser nos IPs créés avec Vivado HLS pour accélérer notre traitement de l'image. En fin de compte, tous de l'image originale, l'image Sobel ... écrira directement à la RAM qui sera utilisé pour le traitement de logiciels et affiché en même temps dans un écran.

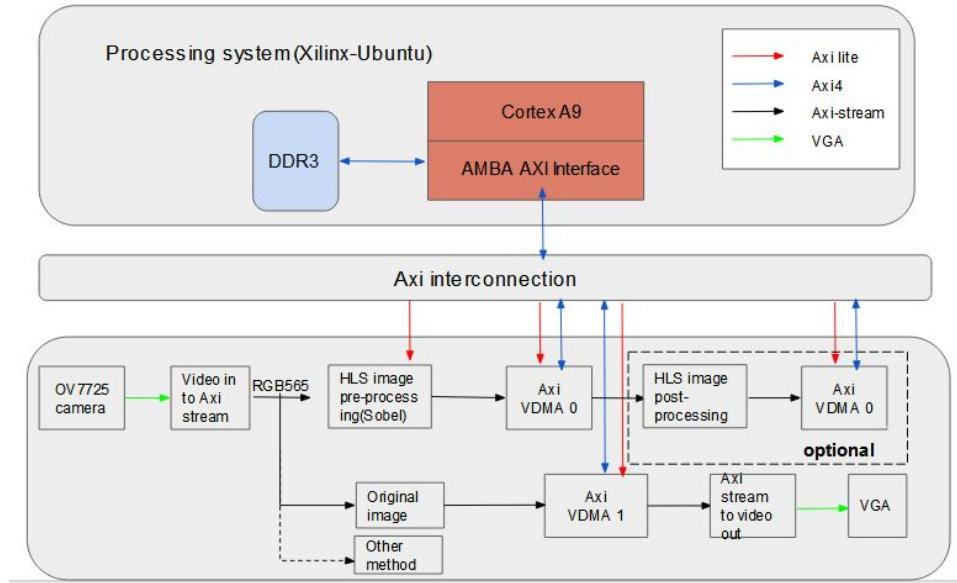


Figure 14. Chaine de traitement d'image

4.1.1 Pilote caméra

Actuellement, dans le domaine de l'acquisition de l'image industrielle, les gens ont utilisé deux capteurs d'image CCD et capteurs d'image CMOS. Sortie du CCD avec le signal général analogique standard, vous avez besoin pour passer à travers le décodeur vidéo pour le contrôleur de signal numérique entrant, et capteurs d'image CMOS signaux numériques directement sortie peut être connecté directement au contrôleur. Avec l'amélioration du niveau technique et technologique de la conception de circuits intégrés, la quantité et les cellules capteur d'image CMOS de pixels de vitesse d'acquisition sont de plus en plus. Parce que des dispositifs à grande vitesse CMOS au cours des dernières années, un nombre croissant de système d'acquisition d'image haute vitesse utilise un capteur d'image CMOS que les dispositifs d'acquisition d'image^[6].

Capteurs CMOS marques et les options couramment utilisées:

Sony: Nissan CMOS haute sensibilité et faible bruit, l'accent mis sur la photographie pour prendre des photos, mais les matériaux de référence relativement moins développés;

Aptina: CMOS conception du système en photographiant l'architecture, le processus de développement est plus compliqué, et le prix est élevé, le prix est légèrement plus élevé la qualité de l'image;

OmniVision: CMOS architecture du système d'acquisition d'image plus adaptée pour développer une information plus adéquate, une bonne compatibilité entre chaque série

OV7725 CMOS Capteur

Notre objectif est de configurer l'appareil pour travailler en mode VGA565 et dispose d'une sortie de signal correct avec une fréquence de 25 Mhz. Tout d'abord, nous devons comprendre la caractéristique principale de OV7725, les signaux, etc.

Tableau 4. caractéristique principale du OV7725^[7]

Array Size		640 x 480
Power Supply	Digital Core	1.8VDC + 10%
	Analog	3.0V to 3.6V
	I/O	1.7V to 3.3V
Output Format	8-bit	• YUV/YCbCr 4:2:2
		• RGB565/555/444
		• GRB 4:2:2
		• Raw RGB Data
	10-bit	• Raw RGB Data
	Max Image Transfer Rate	60 fps for VGA

OV7725 ont de nombreux broches, mais le système utilisé dans le module OV7725 ne contient que certains des broches suivantes:

Tableau 5. Les broches du OV7725^[7]

Nom	Type de Broche	Description	Design
D0~D9	Output	Data output bit[0-9]	utiliser que D2-D9
RESET	Input	System reset, active '0'	toujours mettre '1'
PWDN	Input(0)	Power Down Mode Selection	-
PCLK	Output	Pixel clock output	50Mhz
XCLK	Input	System clock input	25Mhz
HREF	Output	HREF	-
VSYNC	Output	Vertical sync	-
SIOC	Input	SCCB serial interface clock	-
SIOD	I/O	SCCB serial interface data I/O	-

Parce que le nombre de broches utilisés est pas beaucoup, donc nous avons choisi d'utiliser deux interfaces de PMOD de ZYBO JB et JE pour connecter avec OV7725. Le pilote OV7725 contient deux actions: 1. Configuration des registres 2. Combinaison des données entrantes pour avoir pixel RVB selon les signaux entrants (PCLK, HREF, VSYNC).

1. Configuration des registres

La configuration des registres sont conformes avec le protocole I2C. Dans cet exemple, nous avons construit un module maître d'I2C, et pour la configuration, il faut seulement deux instructions:

0x1100; // 11 est CLKRC registre, la valeur est réglée sur 00, l'horloge interne

0x1206; // définir la résolution VGA 640x480, format de sortie de pixel est fixé à RGB565

Tableau 6. Registres du OV7725^[7]

Address (Hex)	Register Name	Default (Hex)	R/W	Description
10	AEC	00	RW	<p>Exposure Value</p> <p>Bit[7:0]: AEC[7:0] (see register AECH for AEC[15:8]) AEC[15:0] = {AECH[7:0] (0x08), AEC[7:0] (0x10)} $T_{\text{exposure}} = \text{AEC}[15:0] \times T_{\text{row interval}}$</p>
11	CLKRC	00	RW	<p>Internal Clock</p> <p>Bit[7]: Reserved</p> <p>Bit[6]: Use external clock directly (no clock pre-scale available)</p> <p>Bit[5:0]: Internal clock pre-scalar $f_{\text{internal clock}} = f_{\text{input clock}} \times \text{PLL multiplier} / [(CLKRC[5:0] + 1) \times 2]$</p>
12	COM7	00	RW	<p>Common Control 7</p> <p>Bit[7]: SCCB Register Reset 0: No change 1: Resets all registers to default values</p> <p>Bit[6]: Resolution selection 0: VGA 1: QVGA</p> <p>Bit[5]: BT.656 protocol ON/OFF selection</p> <p>Bit[4]: Sensor RAW</p> <p>Bit[3:2]: RGB output format control 00: GBR4:2:2 01: RGB565 10: RGB555 11: RGB444</p> <p>Bit[1:0]: Output format control 00: YUV 01: Processed Bayer RAW 10: RGB 11: Bayer RAW</p>

2. Combinaison des données entrantes pour avoir pixel RVB

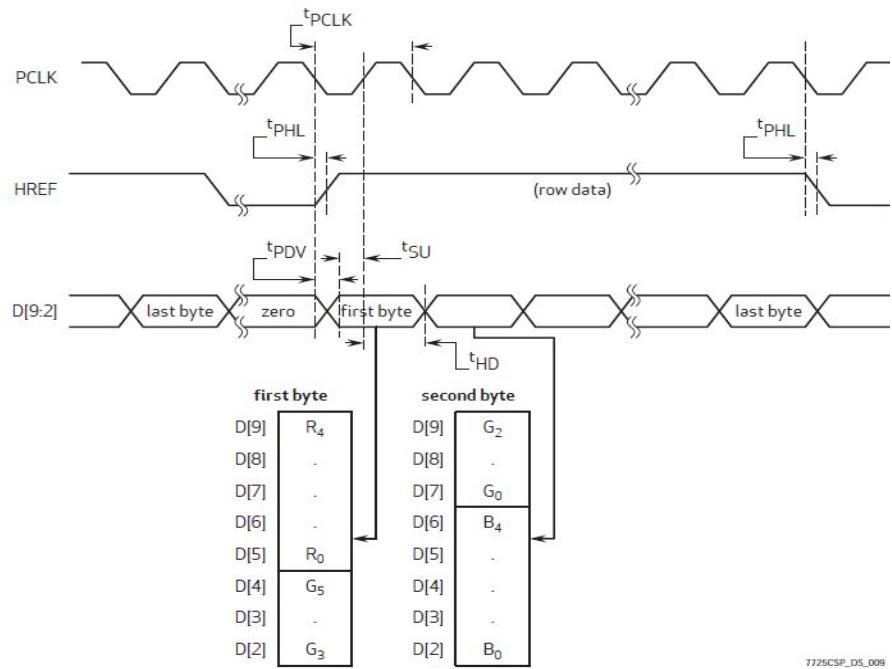


Figure 15. RVB565 VGA chronogramme^[7]

Lorsque VSYNC est valid et HREF est élevé, le première front montant de PCLK va lire le premier octet ($D_7 \sim D_0$). Cette fois-ci à noter que cela ne représente pas le premier octet de pixel, mais les bits de R [4: 0] et de G [5: 3] du premier pixel, l'octect lit par le deuxième front montant de PCLK est les bits de G [2: 0] et de B [4: 0] du premier pixel. Lorsque l'arrivée de la deuxième PCLK hausse, ces deux octets combinés en un pixel complet, nous obtenons le premier pixel. Ainsi, la collecte de ses données (640x2 données), vous obtenez 640 valeurs de pixels. Lorsque complété l'acquisition de 480 lignes, nous finissons l'acquisition d'une trame de données.

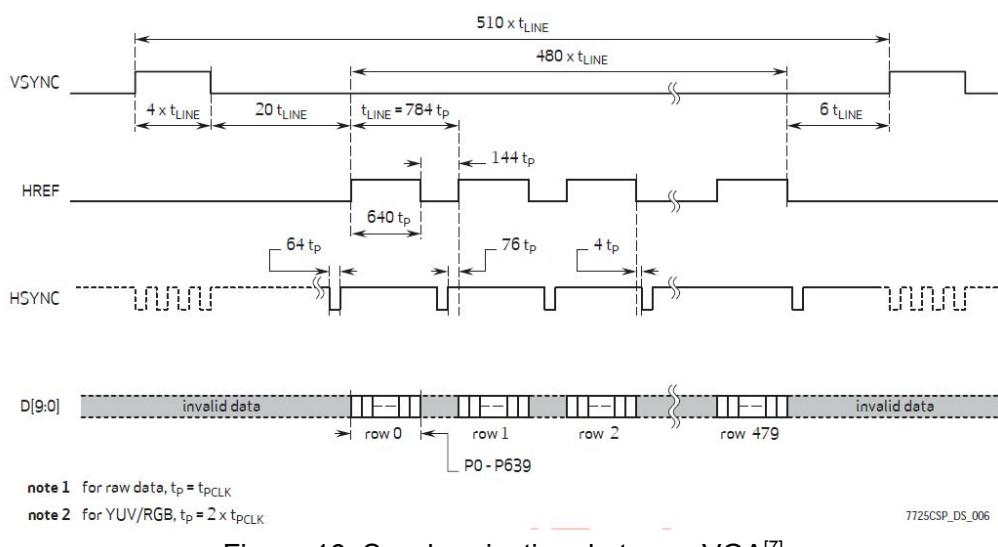


Figure 16. Synchronisation de trame VGA^[7]

Selon la chronogramme d'OV7725 VGA , chaque ligne a 640x2 un valable PCLKs, 144x2 invalide PCLKs, donc chaque ligne dépenses 784x2 PCLKs, et chaque trame a 510 lignes donc 480 lignes sont valides. Par conséquent, un temps d'acquisition de données est $784 * 510 * 2$ PCLKs.

Mais, normalement pour le l'affiche VGA, il support quelques formats Tableau 7. Il n'existe pas 784 * 510, si nous voulons donc afficher sur l'écran avec synchronisation il faut stocker l'image entière et faire une transformation de résolution qui va utiliser beaucoup d'espace de SRAM dans le chip. Il est une mauvais choix et une image de couleur entière a 640 * 480 * 3 octets, égale 900KB même si une image de niveau gris qui a 300KB, la taille de mémoire est trop grand pour ZYBO Tableau 2. En fin, il est une mauvais idée et il est aussi impossible.

Tableau 7. Résolutions populaires pour VGA

Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25

Finalement, parce que le DRAM de ZYBO a une taille de 512 MB, nous pouvons premièrement stocker l'image dans le DRAM et l'afficher après.

4.1.2 Chaîne de vidéo

Nous pouvons utiliser l'avantage d'architecture de ZYNQ pour créer un chemin ou une chaîne de traitement d'image out de vidéo. L'entreprise Xilinx fournit des IPs utils pour l'utilisateur. Dans notre design, nous allons utiliser trois IP principle, VDMA, Video In to AXI4-Stream et AXI4-Stream to Video Out qui utilisent le protocole AXI-Stream. AXI4-Stream est une protocole pour transférer le vidéo plus facilement. Les signaux d'AXI4-Stream sont aux-dessous(Tableau 8).

Tableau 8. Liste des signaux d'AXI-Stream^[8]

Signal	Source	Description
ACLK	Clock source	The global clock signal. All signals are sampled on the rising edge of ACLK .
ARESETn	Reset source	The global reset signal. ARESETn is active-LOW.
TVALID	Master	TVALID indicates that the master is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
TREADY	Slave	TREADY indicates that the slave can accept a transfer in the current cycle.
TDATA[8n-1]:0]	Master	TDATA is the primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
TSTRB[(n-1):0]	Master	TSTRB is the byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte.
TKEEP[(n-1):0]	Master	TKEEP is the byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream. Associated bytes that have the TKEEP byte qualifier deasserted are null bytes and can be removed from the data stream.
TLAST	Master	TLAST indicates the boundary of a packet.
TID[(i-1):0]	Master	TID is the data stream identifier that indicates different streams of data.
TDEST[(d-1):0]	Master	TDEST provides routing information for the data stream.
TUSER[(u-1):0]	Master	TUSER is user defined sideband information that can be transmitted alongside the data stream.

Il existe deux signaux très importantes d'AXI4-Stream, **TLAST** ou **EOL** et **TUSER** ou **SOF**. Le signal d'**EOF**, physiquement transmis sur le AXI4-Stream ‘**TLAST**’ signal marque le dernier pixel d'une ligne. L'impulsion est **EOL** 1 transaction valide de large, et doit coïncider avec le dernier pixel d'une ligne de balayage.

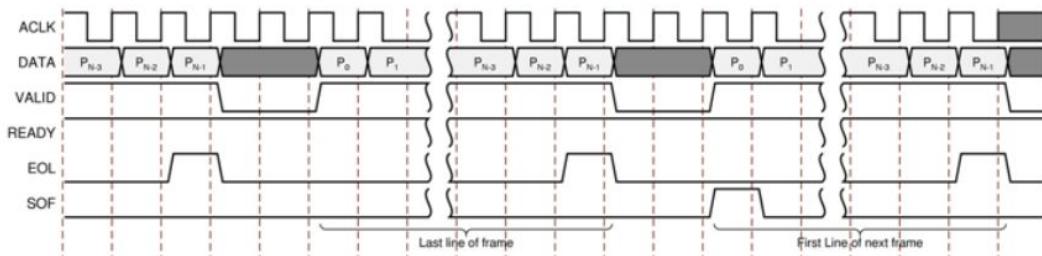


Figure 17. Utilisation des signaux d'EOL et SOF^[8]

Pour gérer le vidéo de caméra OV7725, il est facile et nous nous profitons beaucoup du protocole AXI4-Stream, nous avons donc adopté ce protocole et avons fait des IPs standards. Mais, puisque notre signaux du vidéo venu de OV7725 sont VSYNC et HSYNC, il faut le changer à protocole AXI4-Stream. L'entreprise Xilinx a déjà fourni ce IP ‘Video In to AXI4-Stream’ et ‘AXI4-Stream to Video Out’ pour transformer les signaux de caméra au AXI4-Stream. ‘Video In to AXI4-Stream’ transfert les données vidéo au type AXI4-Stream. Dans ce projet, les données vidéo viennent du module de caméra, la caméra ne doit fournir que quatre signaux: synchronisation horizontale, synchronisation verticale, les données de pixel de 24(8bit)bits signal valide. Pour l'IP AXI4-Stream to Video Out’ qu'il a une fonction l'inverse de ‘Video In to AXI4-Stream’.

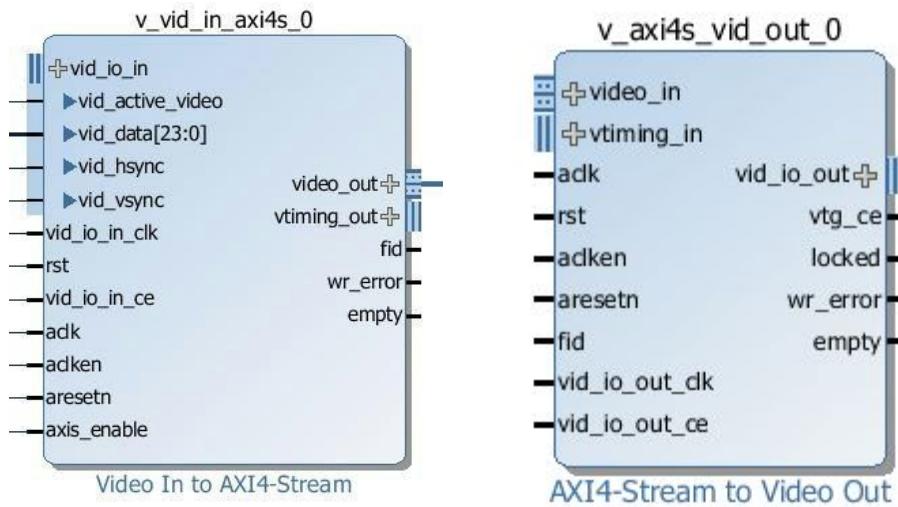


Figure 18. IPs de ‘Video In to AXI4-Stream’ et ‘AXI4-Stream to Video Out’

Le plus important du design de la chaîne est le VDMA qui s'adapte bien au protocole AXI4-Stream. En fait, VDMA est une module DMA qui a haut vitesse d'accès au stockage spécifiquement pour le flux de données vidéo. Il reçoit des données vidéo via le protocole AXI-Stream mais le signal de contrôle (telles que la taille de la mémoire tampon de trame, ouverture et fermeture de DMA, ainsi que d'autres paramètres) est accessible à travers AXI-Lite à partir d'autres interfaces. VDMA a deux chemin de DMA, le chemin de S2MM AXI4-Stream écrit flux de données à la mémoire tampon de trame spécifié. Au contraire, MM2S lit tampon de trame et le transforme au flux de données qui respecte AXI4-Stream.

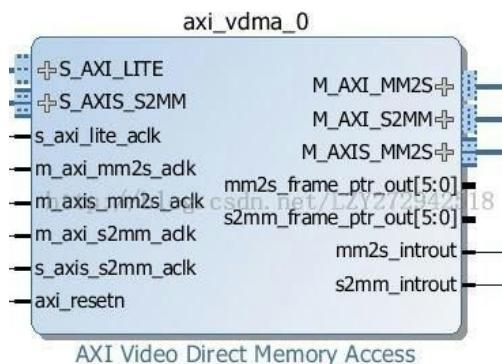


Figure 19. Module VDMA

Pour la VDMA, nous avons besoin de le configurer avec un tampon de frame d'au moins trois de pour éviter les conflits. Nous avons donc mis 3 frame buffers.

D'ailleurs, pour accélérer le process de traitement d'image dans logiciel, nous allons stocker deux image différent en même temps avec deux chemins séparés, un stocke l'image origine, l'autre stocke l'image qui contient le bord grâce à le module sobel.

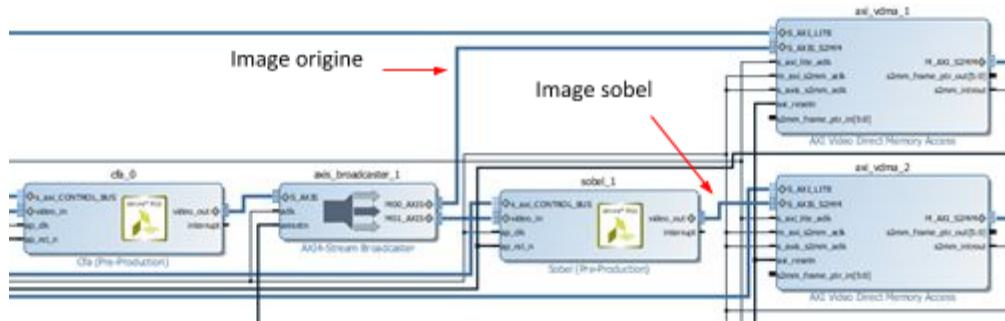


Figure 20. deux chemins séparés

4.1.3 Format d'image - Bayer Raw ou RGB 565

Comme j'ai dit dans la section de Caméra OV7725, il a deux format d'image que nous pouvons adopter 'bayer raw' et 'RGB 565', il est facile d'utiliser RGB 565 et la qualité d'image est assez bonne pour le traitement suivant. Mais, à l'intérieur de caméra de notre design, la fréquence de génération des pixel est deux fois de la fréquence entré(25 MHz) – 50MHz (PCLK) et avec notre connection des fils très proche, il y a des distorsion pour la pin PCLK.

Si nous utilisons l'image bayer raw, il faut jouter l'IP (CFA : Color Filter Array Interpolation) pour faire une interpolation linéaire normalement. Finalement, nous avons trois choix :

(a) Utilisation direct de format RGB 565

Il est simple de le faire, le flux de vidéo est montré dans la figure 9(a).

(b) Utilisation le format de bayer raw et développer un sous-module CFA dans le driver(b)

(c) Utilisation le format de bayer raw et développer un CFA fourni par Xilinx qui il faut acheter(c)

(d) Utilisation le format de bayer raw et développer un CFA développé par nous même (d)

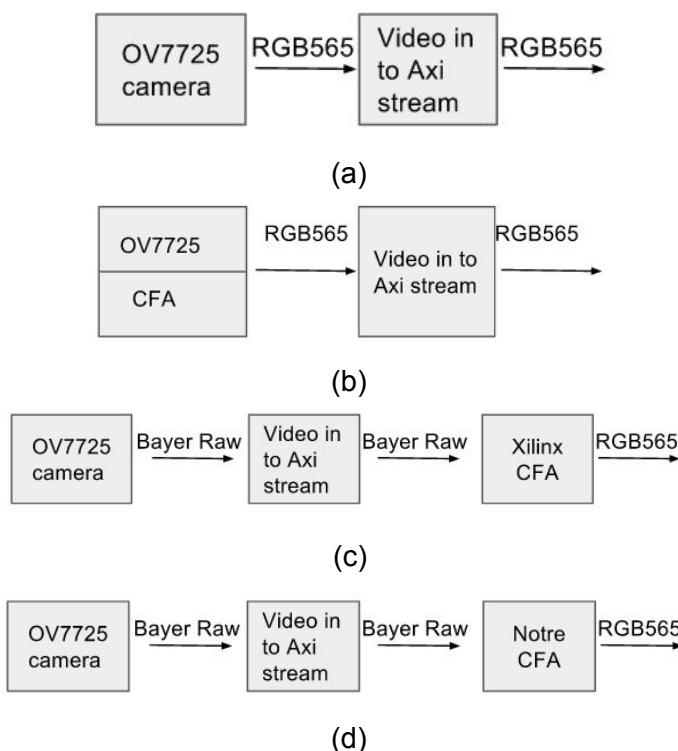


Figure 21. choix pour différents formats

Si nous choisissons le format RGB565 directement, il est simple à réaliser mais nous perdons la qualité d'image. De plus, nous avons utilisé des lignes Dupont pour connecter le caméra et la carte et le caméra tourne avec 50 Mhz intérieur quand il envoie format RGB565, nous avons donc le problème de distorsion entre les fils(intégrité de signal).

Cependant, quand nous adopté la deux façon – créer un sous-module CFA dans driver en utilisant les signaux de synchronisation. Il est traditionnel et tous les pixels sont synchronisés. De même, il a le problème de flexibilité si nous voulons le utiliser comme un module en respectant le protocole AXI4-Stream et il va prendre un peu de temps pour développement.

Finalement, nous voulons créer cet IP en utilisant protocole AXI4-Stream et acheter un IP chez Xilinx n'est pas agréable pour nous, nous voulons créer ce module nous même.

4.2 Système de contrôle des périphériques

4.2.1 Motivation

Étant la fiabilité la caractéristique la plus importante pour le design des systèmes embarqués avec champ d'application critique (comme l'avionique/l'automobile), quelques stratégies sont classiquement adoptées dans la conception des produits sûrs:

- Redondance: plusieurs composants matériels qui travaillent en parallèle pour assurer le fonctionnement continu du système.
- Découplage: fonctionnement indépendant des parties critiques pour isoler les "bugs".

Vue la motivation de ce projet - "conception d'une voiture intelligente capable de détecter des obstacles au travers du traitement d'image" - nous avons décidé de mettre en oeuvre un découplage entre le système de traitement d'image et le système de contrôle des périphériques (gestion du mouvement de la voiture, des capteurs et des modules de communication).

Le but principal de cette partie est, donc, d'implémenter un système d'arrêt d'urgence qui soit indépendant, ainsi qu'une interface utilisateur que puisse toujours contrôler la voiture quoi que se passe avec le système de traitement d'image.

4.2.2 Architecture

La distribution de ces deux systèmes dans la carte utilisée (Zybo Zynq-7000) est faite de la manière suivante:

- Partie PS ("Programable System"): S'occupe du contrôle du flux vidéo de la caméra (contrôle des DMA's) et aussi du traitement d'image.
- Partie PL ("Programmable Logic"): Accélération des algorithmes de traitement d'image en matériel et gestion des périphériques au travers d'un "soft core" (MicroBlaze).

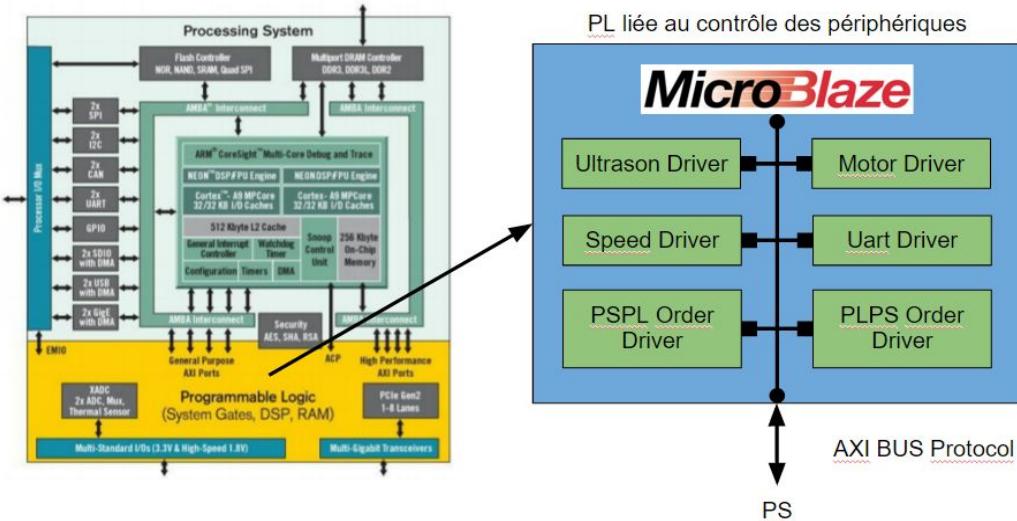


Figure 22. MicroBlaze

On va se concentrer sur cette session, sur la partie PL - MicroBlaze et périphériques.

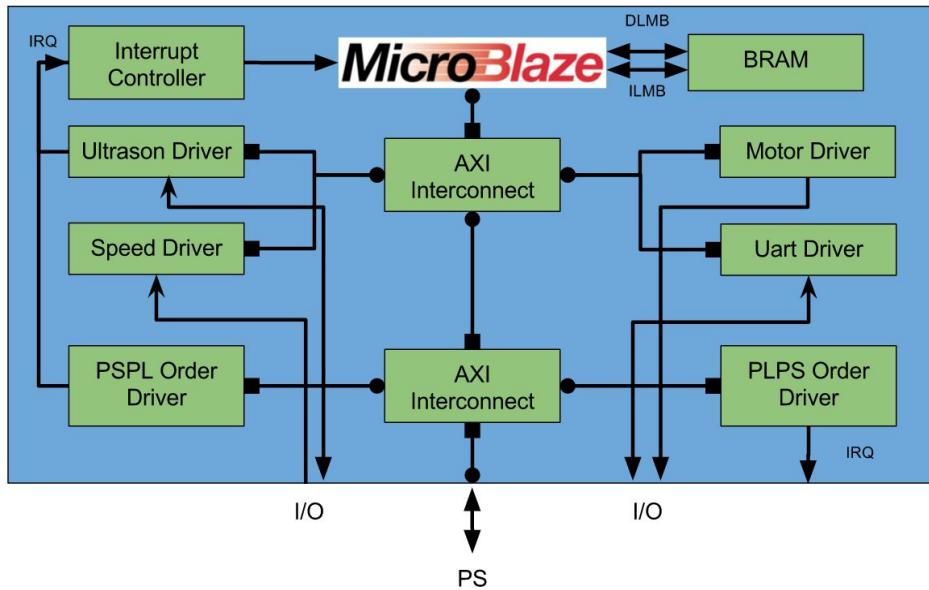


Figure 23. Périphériques de MicroBlaze

4.2.2.1 Microblaze

C'est un “soft core” disponible dans la bibliothèque standard des IP's Xilinx qui peut être configuré à plusieurs niveaux de complexité. La stratégie que nous avons gardé dans le choix de ses paramètres a été:

- Avoir un système de gestion d'interruptions.
- Parallélisme et temps réel de la gestion des périphériques traduit sur des modules programmés en matériel se communiquant avec le coeur MicroBlaze.

Si dessous le code TCL utilisé pour l'intégration du module à notre “block design” sur Vivado 2015.2:

```

create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze:9.5 microblaze_0
apply_bd_automation -rule xilinx.com:bd_rule:microblaze -config {local_mem "128KB" ecc
"None" cache "8KB" debug_module "Debug Only" axi_periph "Enabled" axi_intc "1" clk
"/processing_system7_1/FCLK_CLK2 (50 MHz)" } [get_bd_cells microblaze_0]
set_property -dict [list CONFIG.G_USE_EXCEPTIONS {1}] [get_bd_cells microblaze_0]

```

Remarque: Dans la version de développement du projet on a eu besoin d'un bloc mémoire de taille trop élevée pour le MicroBlaze (128KB). Ça c'est du au fait que nous utilisions des bibliothèques pas forcément nécessaires à un cas d'utilisation normal (sans besoin de messages de "print" pour le "debug", par exemple). La taille standard lors de la création d'un bloc mémoire local pour MicroBlaze c'est de 8KB.

4.2.2.2 Modules AXI (Périphériques)

Vivado 2015.2 offre la possibilité de créer des nouvelles IP's en s'appuyant sur un "wrapper" AXI4. Au travers de cette méthode, on choisit un numéro de registres internes, et ensuite toute l'interface d'accès à ces registres est mise en oeuvre en respectant le protocole de bus AXI. La logique programmée par nous est donc, rajoutée au module pour interagir avec les registres et implémenter la fonctionnalité désirée.

4.2.2.2.1 Contrôle des moteurs

Ce module est chargé de contrôler la petite carte pont H liée aux quatre moteurs de la voiture. La possibilité d'avoir des différentes vitesses est aussi prise en considération une fois que les sorties de ses modules sont en vrai, de sorties en PWM configurables.

Registres AXI:

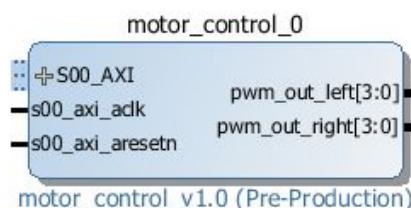


Figure 24. IP de motor

slv_reg0: Pwm period left (clock_ticks)
 slv_reg1: Pwm period right (clock_ticks)
 slv_reg2: Duty cycle left (clock_ticks)
 slv_reg3: Duty cycle right (clock_ticks)
 slv_reg4: Motors Control

Logique implémentée:

- Mise en oeuvre des compteurs avec comparaison entre les valeurs sur les quatre premiers registres afin de simuler le comportement des PWM's pour "pwm_out_left" et "pwm_out_right".
- Habiliter "pwm_out_left" égal à slv_reg4[9] = 1.
- Valeur attribuée à "pwm_out_left" pendant le "duty cycle" égal à slv_reg4[7:4].

- Habiliter “pwm_out_right” égal à slv_reg4[8] = 1.
- Valeur attribué à “pwm_out_right” pendant le “duty cycle” égal à slv_reg4[3:0].

Remarque: Les valeurs attribuées à “pwm_out_left” et “pwm_out_right” sont normalement parmi 0xA/0x5/0x0 pour une marche “toute droite”/“en arrière”/“stop”. Ces valeurs ne sont pas “hard codées” sur le module parce que l’ordre des fils de contrôle des roues aurait pu être altéré ou même le module pont H utilisé aurait pu être changé.

4.2.2.2 Capteur de distance ultrason

Ce module est chargé de contrôler le module HC-SR04 ultrason. Son schéma de fonctionnement est affiché dans la figure ci dessous. “Trigger” c'est l'input du module et gère l'émission du signal de “scan” ultrason. “Echo” c'est l'output du module HC-SR04 et indique le temps écoulé entre l'émission et la réception des ondes sonores envoyées précédemment par l'activation du “Trigger”.

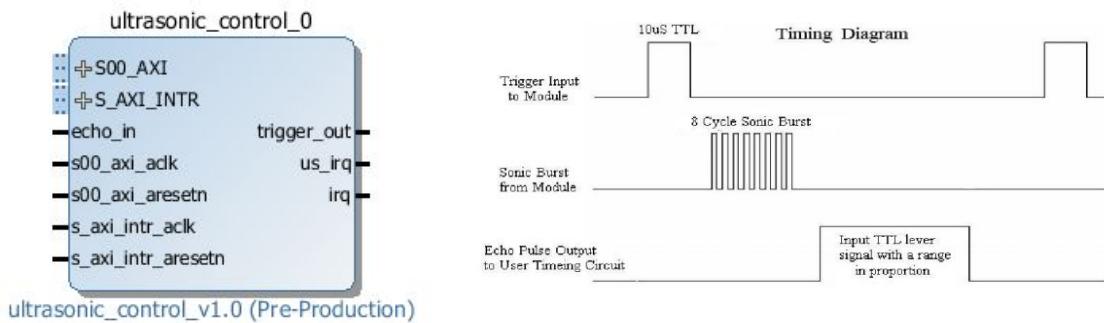


Figure 25. IP d’ultrason

Registres AXI:

- slv_reg0: Trigger high level (clock_ticks)
- slv_reg1: Measure cycle (clock_ticks)
- slv_reg2: Min distance (cm) -> Max Intern Counter (clock_ticks)
- slv_reg3: Min valid distance (cm) -> Max Valid Intern Counter (clock_ticks)
- slv_reg4: IRQ tolerance/ Last measured time (clock ticks)

Logique implémentée:

- Mise en œuvre d'une machine à état basé sur les états Trigger_On/Wait_Echo/Echo_On/Wait_Cycle.
- Des compteurs pour la durée des pulses, pour le cycle de mesure et pour le nombre d'interruptions consécutives.
- Lors d'une transition négative sur “echo_in” sans “timeout”, comparaison avec “distance minimale valide” et “distance minimale permise”. Si valide, la distance est stockée dans un registre interne accessible (à l'extérieur) sur l'adresse de lecture de “slv_reg4”, sinon, le “compteur d'interruption consécutives” est incrémenté, possiblement entraînant une interruption.

Calcul de distance:

Compteur = I;
Clock = C;

```

Distance = D;
D = ((C - I) *17000)/C;

```

Remarque: La constante 17000 veut dire 17000 cm/s (ou 34000/2 cm/s) puisque la distance mesurée est en fait deux fois la distance réelle (temps d'aller-retour des ondes sonores).

4.2.2.2.3 Capteur de vitesse

Ce module est chargé de capter les signaux prévenants des capteurs de vitesse. Dès qu'un tournage des roues est identifié il garde une nouvelle valeur pour la vitesse de chaque paire de roues à gauche et à droite.

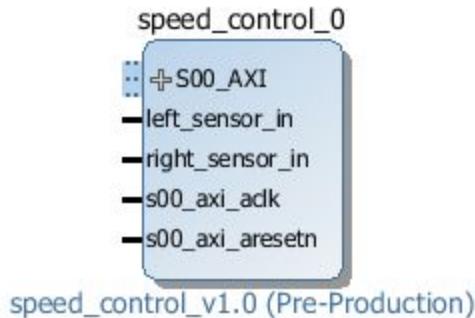


Figure 26. IP de vitess

Registres AXI:

slv_reg0: Spared cycles Left (clock_ticks)

slv_reg1: Spared cycles Right (clock_ticks)

slv_reg2: Module Enable

Logique implémentée:

- Mise en œuvre des compteurs sur des transitions positives sur "left_sensor_in" et "right_sensor_in".
- Stockage du nombre de cycle dépassés après 20 transitions positives sur "left_sensor_in" et "right_sensor_in". (Cela correspond au numéro des divisions de la structure circulaire du capteur)

Calcul de vitesse:

```

Spared cycles = S;
Clock = C;
Distance for 1 turn = D;
D = 2 *π *3.25;
Speed = (S/C) *D = (S * 2 *π *3.25)/(50 *10**6);

```

4.2.2.2.4 Module de communication PSPL

Ce module est chargé d'implémenter la communication entre la partie ARM et la partie MicroBlaze. L'interruption générée par ce module doit donc être relié au système chargé de lire sur ce module (celui qui effectuera juste de lectures et jamais des écritures).

Figure 27. IP de communication

Registres AXI:

- slv_reg0: Message
- slv_reg1: Parameters
- slv_reg2: Parameters
- slv_reg3: Parameters
- slv_reg4: Generate IRQ

Logique implémentée:

- Génération d'une interruption sur un cycle d'horloge après écrire la valeur "1" (un) sur slv_reg4.

Remarque: Les modules UART et de Contrôle d'interruption sont déjà fournis dans la bibliothèque d'IP's sur Vivado 2015.2.

4.2.3 Programme

Comment le programme qui tourne sur MicroBlaze est organisé globalement:

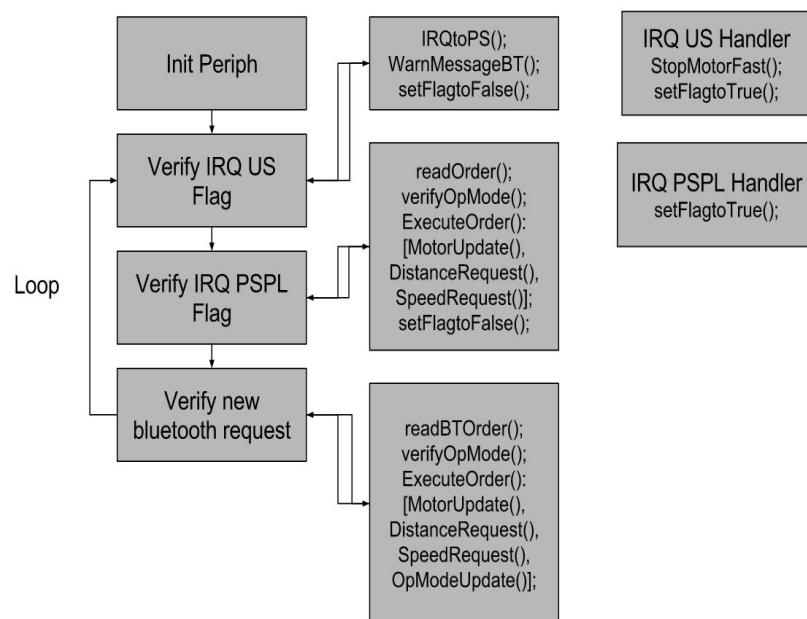
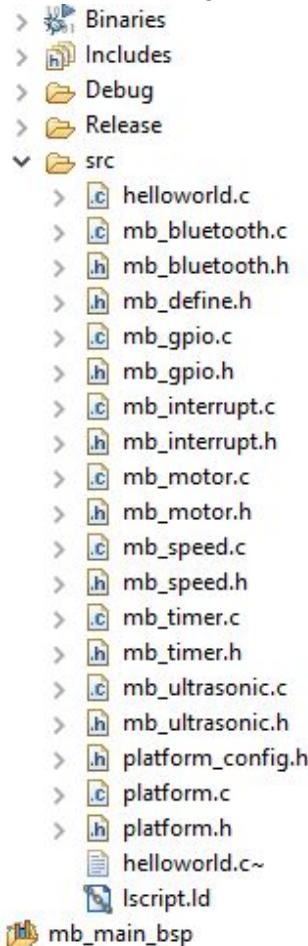


Figure 28. Boucle principale

Initialisation:

Tout le périphériques sont configurés et initialisés.

Vérification de “flags” d'interruption:

Les messages d'erreur vers les autres systèmes (partie ARM ou smartphone) sont moins prioritaires que l'arrêt total de la voiture face à un obstacle trop proche, par exemple. Donc les “handlers” d'interruption ne font que le minimum et strictement nécessaire. Pour le ultrason, il arrête la voiture immédiatement et met son “flag” correspondant à “true” pour qu'après dans la boucle principale, des messages d'erreur soient envoyées. Pour le module PSPL il met son “flag” correspondant à “true” pour que la voiture puisse lire et exécuter ses instructions dans son prochain passage par la boucle principale.

Verify bluetooth request:

Essaye de lire un caractère provenant de l'interface Bluethooth sur UART. S'il réussit, il garde la commande (si valide) et l'exécute si c'est la dernière (d'après les tailles connus pour l'échange de messages entre les différents systèmes décrits dans la prochaine session)

Génération du BOOT.BIN:

Pour que le programme soit véritablement chargé sur la mémoire locale du Microblaze lors du démarrage de la carte, il faut créer un BOOT.bin, un archive concentrant le programme “bootloader” de la carte, le “bitstream” du système et, dans notre cas (où linux est en place), le u-boot.elf (programme de chargement du système d'exploitation sur la carte).

- 1) Même sans avoir la carte (Zybo) connectée, sélectionner l'option “Program FPGA” sur le menu d'outils Xilinx et sélectionner le programme dédié au MicroBlaze sur “ELF/MEM file to initialize in Block RAM”. Après, tapez sur “Program”. Cela va générer un download.bit sur le fichier hardware de votre projet SDK.
- 2) Sur le menu d'outils Xilinx, chercher “Create Zynq Boot Image”. Sélectionner l'option “Create new BIF file” (si c'est la première fois que vous générez votre BOOT.bin) et sélectionner le répertoire pour sauvegarder le “schéma” de votre image de boot (pour ne pas avoir besoin de tout sélectionner dans une prochaine génération de l'image). Sélectionner l'exécutable FSBL (si vous n'en avez pas, il suffit de le régénérer automatiquement sur SDK en sélectionnant comme processeur cible un cœur ARM). Sélectionner le download.bit (“bitstream” avec votre application qui a été généré lors de l'étape 1). Sélectionner le u-boot.elf que vous avez compilé ou que vous avez téléchargé chez Xilinx.

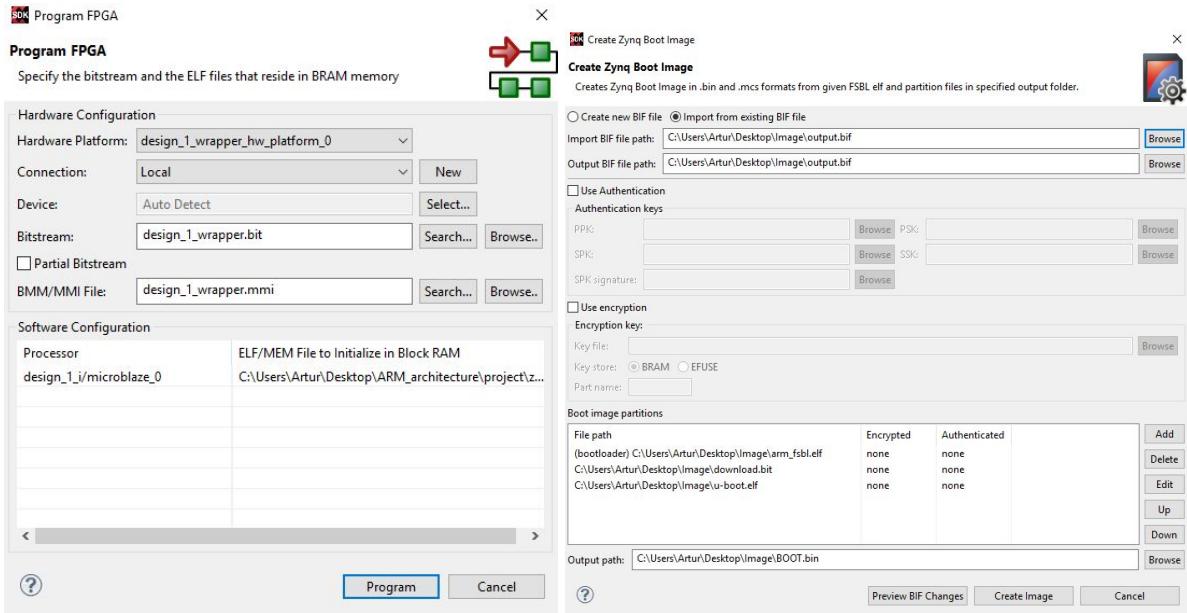


Figure 29. Création de BOOT.bin

4.2.4 Protocole de communication

Ayant besoin de bien définir les messages échangées entre les trois parties de notre projet (partie PS Linux, partie PL Microblaze et application mobile), nous avons défini quelques messages standard pour permettre la communication entre PL/PS et PL/Android.

Communication Smartphone x MicroBlaze FPGA (Core) - Bluetooth sur protocole UART

Requêtes:

W[0-1][0-1][0-9][0-9]

Contrôle des roues:

[0-1] : Tout droit/arrière pour le roues de la gauche

[0-1] : Tout droit/arrière pour le roues de la droite

[0-9] : Vitesse pour le roues de la gauche

[0-9] : Vitesse pour le roues de la droite

S

Requête d'une message de statuts de vitesse

D

Requête d'une message de statuts de distance

M[0-2]

Modifie le mode d'opération de la voiture:

[0] : Contrôle complet de l'utilisateur

[1] : Contrôle complet de la partie traitement d'image

[2] : Traitement d'image corigeant l'utilisateur

Messages:

S[0-1][0-9][0-9][0-9][0-9][0-9][0-9]

Message de statuts de vitesse

[0-1] : Normal/Message d'erreur

[0-9][0-9][0-9] : Vitesse des roues de la gauche en cm/s

[0-9][0-9][0-9] : Vitesse des roues de la droite en cm/s

D[0-1][0-9][0-9]

Message de statuts de distance

[0-1] : Normal/Message d'erreur

[0-9][0-9][0-9] : Distance en centimètres

Communication ARM (Core) x MicroBlaze FPGA (Core) -

Registres 4 x 32 bits accédés par protocole AXI (Messages coté ARM)

Registres 4 x 32 bits accédés par protocole AXI (Messages coté MicroBlaze)

Synchronisation faite par des interruptions

R0 = Message

R1:R3 = Paramètres

PL -> PS:

Requêtes:

0x00000000 = nouveau mode d'opération

Paramètres :

R1 = nouveau mode d'opération

Messages:

0x00000001 = direction des moteurs

Paramètres:

R1[1] = Orientation des moteur de la gauche

R1[0] = Orientation des moteur de la droite

R2 = Vitesse des moteur de la gauche

R3 = Vitesse des moteur de la droite

0x00000002 = collision du module ultrason

Paramètres :

None

0x00000003 = Distance ultrason

Paramètres :

R1 = Distance

0x00000004 = Vitesse

Paramètres :

R1 = Vitesse des roues de la gauche

R2 = Vitesse des roues de la droite

PS -> PL:

Requêtes:

0x00000001 = requête d'une nouvelle direction des moteurs

Paramètres:

R1[1] = Orientation des moteur de la gauche

R1[0] = Orientation des moteur de la droite

R2 = Vitesse des moteur de la gauche

R3 = Vitesse des moteur de la droite

0x00000003 = requête de distance ultrason

Paramètres :

None

0x00000004 = requête de vitesse

Paramètres :

None

5. Linux driver

Maintenant, nous avons le flux binaire de l'ensemble de notre projet qui comprennent les partie de traitement d'image et de contrôle de mouvement. Mais ceci est seulement la conception du matériel et nous avons besoin d'ajouter la commande pour contrôler ces périphériques de bien travailler. En outre, parce que nous utilisons le système Linux, donc nous avons besoin de créer des pilotes pour ces périphérique à assurer que Linux peut les connaître. Dans mon design, il y a deux drivers pour deux fonctionnalité. Un pour le VDMA, l'autre pour les périphériques qui gèrent la communication entre les deux parties - MicroBlaze et ARM (Figure13 PL2PS, PS2PL).

5.1 Driver de VDMA

La figure 14 montre que tous les données d'image vont être stocké dans la RAM via VDMA et nous voulons savoir bien la fonctionnalité de VDMA pour créer ce type de driver. Le table montre des registres importants pour configurer le VDMA et la figure 31 nous donne l'ordre de bien démarrer le VDMA.

Tableau 9. les registres importants d'écriture^[9]

Address Space	Offset Name	Description
30h	S2MM_VDMACR	S2MM VDMA Control Register
34h	S2MM_VDMASR	S2MM VDMA Status Register
38h	Reserved	N/A
3Ch	S2MM_VDMA_IRQ_MASK	S2MM Error Interrupt Mask Register
40h	Reserved	N/A
44h	S2MM_REG_INDEX	S2MM Register Index
A0h	S2MM_VSIZE	S2MM Vertical Size Register
A4h	S2MM_HSIZE	S2MM Horizontal Size Register
A8h	S2MM_FRMDLY_STRIDE	S2MM Frame Delay and Stride Register
ACh to E8h	S2MM_START_ADDRESS (1 to 16)	S2MM Start Address (1 to 16)

S2MM_VDMACR (S2MM VDMA Control Register – Offset 30h)

This register provides control for the Stream to Memory Map VDMA Channel.

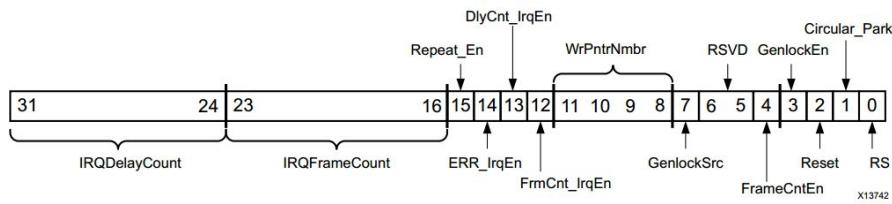


Figure 30. Registre de contrôle de VDMA^[9]

Configure the Write channel registers as follows.

- Set S2MM_VDMACR (30h) to 8Bh. This enables run/stop, Circular_Park, GenlockEn, and GenlockSrc. In this case, external connections of frame pointers are not required. If Repeat Enable and Interrupt on error is required, set bit 15 and bit 14 of this register.
- Set S2MM_Start_Address 1 (ACh) through S2MM_Start_Address 3 (B4h) to their required locations. These locations can be static (based on maximum frame size) or dynamic (based on actual frame size).
- Set S2MM_FRMDLY_STRIDE (A8h) to the appropriate value. FRMDLY is not applicable for the Dynamic Genlock Master. STRIDE is the number of bytes per line.
- Set S2MM_HSIZE (A4h) to the number of bytes per line.
- Set S2MM_VSIZE (A0h) to the number of lines per frame. VSIZE must be set last and starts the S2MM VDMA transactions.

Figure 31. L'ordre pour configuration des registres d'écriture^[9]

En fait, dans le noyau de Linux choisi par nous support le driver de VDMA (figure 32), si nous voulons l'utiliser juste le choisir quand nous faisons la compilation de noyau. Ensuite, nous devons ajouter le VDMA dans Linux via ‘DeviceTree’.

```

axi_vdma_1: dma@43000000 {
    #dma-cells = <1>;
    compatible = "xlnx,axi-vdma-1.00.a";
    interrupt-parent = <&ps7_scugic_0>;
    interrupts = <0 29 4>;
    reg = <0x43000000 0x10000>;
    xlnx,flush-fsync = <0x1>;
    xlnx,num-fstores = <0x3>;
    dma-channel@43000030 {
        compatible = "xlnx,axi-vdma-s2mm-channel";
        interrupts = <0 29 4>;
        xlnx,datawidth = <0x18>;
        xlnx,device-id = <0x0>;
        xlnx,genlock-mode ;
    }
}

```

```

    xlnx,include-dre ;
};

};

```

Par exemple, le code au-dessus montre que ce VDMA est enregistré avec l'adresse physique de 0x4300_0000, son numéro d'interruption est 29, '4' signifie que le type d'interruption. Il y a trois type:

"0 — Leave it as it was (power-up default or what the bootloader set it to, if it did)

1 — Rising edge

4 — Level sensitive, active high"

Après avoir compilé le 'device tree', nous pouvons trouver l'interruption de VDMA est enregistré dans noyau Linux. La question est pourquoi nous ne utilisons pas ce driver directement? Parce que ce driver il juste fait le configuration ou initialisation de VDMA, il a le problème de synchronisation entre écriture et lecture dans la même adresse. Nous voulons donc modifier ce driver pour réaliser notre objective - ne pas avoir distorsion de la lecture d'image. Il y a deux modules de VDMA mais ils ont la même stratégie.

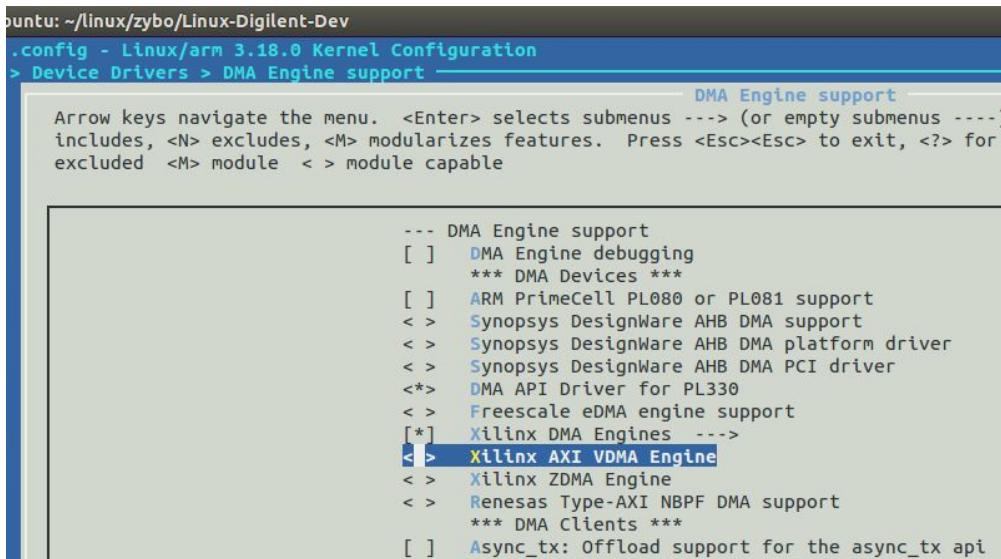


Figure 32. Choix de driver VDMA

5.1.1 Synchronisation avec interruption

Pour réaliser la synchronisation entre lecture et écriture, c'est-à-dire qu'il faut une communication entre lecture et écriture. Le programme ou logiciel de traitement d'image doit savoir quel espace d'adresse qu'il peut lire et annoncer la côté d'écriture pour lui donner l'adresse qui est en train d'être utilisé, donc la côté d'écriture ne peut pas toucher dans cette adresse. De plus, pour assurer le vitesse de programme, logiciel ou lecture ne doit pas attendre l'écriture. En fin, en donnant l'hypothèse que la vitesse de lecture et toujours plus lente qu'en écriture. Nous concevons une idée.

Grâce aux registre 'S2MM_START_ADDRESS' (ACh to E8h), nous pouvons renouveler les valeurs de registres avec l'adresse que nous voulons écrire. Quand une interruption arrive,

c'est-à-dire que le VDMA est fini écrire une image dans l'adresse 0, donc, la carte veut la lire et annoncer le driver que l'adresse 0 est occupé, pour le faire il renouvelle les trois registres pour éviter toucher l'adresse 0. S'il marche c'est bien, mais malheureusement il marche pas parce que quand l'interruption est arrivé, l'écriture dans l'adresse 0 est fini c'est bon, mais l'écriture de l'adresse 1 est déjà commencé qui est le fait que nous ne pouvons pas le changer. Ce que nous pouvons faire est changer la valeur de troisième registre.

Enfin, pour lire l'image correctement, la lecture juste suit l'écriture dans la même zone, et **puisque la vitesse d'écriture est toujours plus rapide qu'en lecture, si lecture démarre un peu après l'écriture, il n'y a pas le problème de conflit. Il faut donc juste deux espaces de frame buffers au lieu de trois frame buffers.**

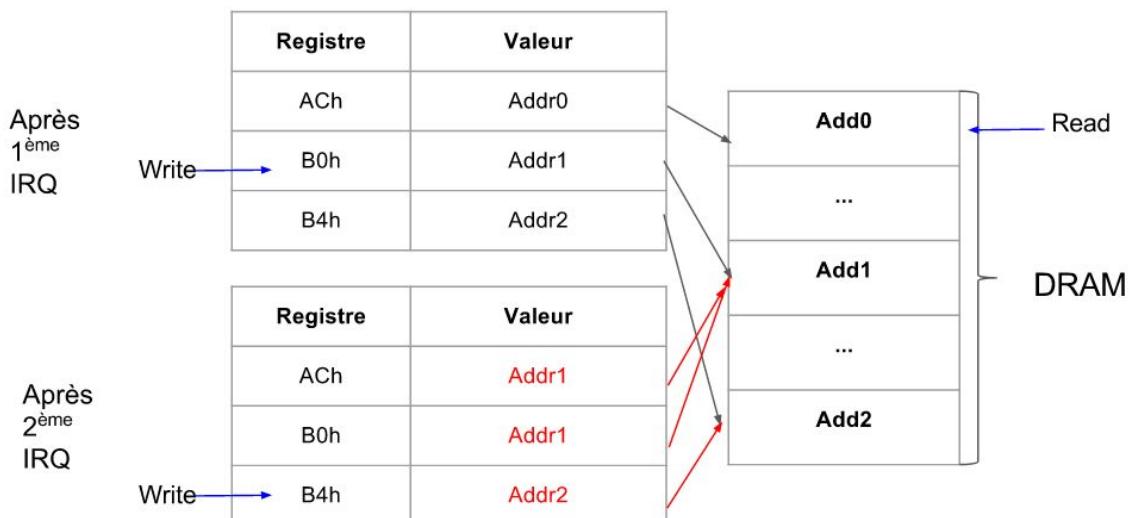


Figure 33. Hypothèse faux de VDMA

5.1.2 Frameworks^[10]

De plus en plus de périphériques ne sont plus implémentés directement comme des périphériques caractères mais utilisent un framework qui permet de factoriser les parties identiques des pilotes contrôlant le même type de périphériques et qui permet d'offrir une interface unique et cohérente aux applications (mêmes paramètres ioctl quel que soit le pilote par exemple). Du point de vue des applications, le périphérique reste vu comme un périphérique caractère normal que son pilote utilise un framework ou pas.

5.1.2.1 Misc framework

Lorsqu'aucun framework ne semble convenir pour un périphérique, il est possible de l'implémenter comme un périphérique caractère directement ou d'utiliser le framework misc qui simplifie cette tâche.

Un périphérique utilisant ce framework est décrit par la structure struct miscdevice (définie dans include/linux/miscdevice.h) :

```
struct miscdevice {
    int minor;
```

```

const char *name;
const struct file_operations *fops;
struct list_head list;
struct device *parent;
struct device *this_device;
const struct attribute_group **groups;
const char *nodename;
umode_t mode;
};


```

Les principaux champs de cette structures sont :

- **minor** : le numéro minor désiré pour le périphérique (le numéro major sera automatiquement celui des périphériques misc, c'est-à-dire 130) ou ISC_DYNAMIC_MINOR pour en obtenir un dynamiquement
- **name** : le nom du périphérique, utilisé pour créer plus ou moins automatiquement le fichier spécial correspondant dans /dev
- **fops** : pointeur vers la structure struct file_operations qui décrit les fonctions utilisées pour répondre aux opérations de lecture, écriture, etc.

Un périphérique s'enregistre en tant auprès du framework misc grâce à la fonction :

```
int misc_register(struct miscdevice *misc);
```

L'appel à cette fonction est traditionnellement effectué dans la fonction probe du pilote.

Lorsque le périphérique n'est plus présent, il faut penser à appeler la fonction :

```
int misc_deregister(struct miscdevice *misc);
```

L'appel à cette fonction est traditionnellement effectué dans la fonction remove.

5.1.2.2 Liens entre les structures

Un pilote, même simple, va avoir besoin de jongler avec de nombreuses structures de données. Nous allons voir dans cette section les liens entre ces structures et comment passer de l'une à l'autre.

Nous allons prendre pour cela un petit exemple de pilote, s'enregistrant auprès du bus platform et utilisant le framework misc pour communiquer avec l'espace utilisateur. Il faut bien garder à l'esprit que plusieurs périphériques physiques peuvent être gérés par un même pilote.

En interne, le pilote fait le choix de représenter toutes les informations utiles pour un périphérique physique grâce à la structure suivante :

```

struct foo_device {
/* Données propres à un périphérique (exemple) */
int irq_num;
int parameter_x;
/* Le périphérique misc correspondant */

```

```

struct miscdevice miscdev;
};

```

Lorsqu'un périphérique est détecté (ici nous avons pris l'exemple du bus platform donc la détection provient soit de la lecture du Device Tree soit de la déclaration statique des périphériques présent lors du démarrage de la plate-forme), la fonction probe est appelée :

```

static int foo_probe(struct platform_device *pdev)
{
    struct foo_device *foodev;
    int ret;
    /* Alloue la mémoire pour une nouvelle structure foo_device */
    foodev = devm_kzalloc(&pdev->dev, sizeof(struct foo_device),
    GFP_KERNEL);
    if (!foodev)
        return -ENOMEM;
    /* Initialise la structure foo_device, par exemple avec les
     informations issues du Device Tree */
    foodev->irq_num = ....;
    foodev->parameter_x = ....;
    /* Initialise la partie miscdevice de foo_device */
    foodev->miscdev.minor = MISC_DYNAMIC_MINOR;
    foodev->miscdev.name = "fooX";
    foodev->miscdev.fops = &foo_fops;
    foodev->miscdev.parent = &pdev->dev; /* (1) */
    platform_set_drvdata(pdev, foodev); /* (2) */
    8
    /* S'enregistre auprès du framework misc */
    ret = misc_register(&foodev->miscdev);
    /* ... */
}

```

Deux liens entre les structures sont mis en place ici :

1. Un lien depuis l'instance de la structure struct *platform_device* vers l'instance de la structure foodev grâce à la fonction ***platform_set_drvdata*** (voir (2)). La fonction ***platform_get_drvdata*** peut être utilisée ultérieurement pour récupérer l'instance de la structure foodev. Ce lien sera utilisé par exemple dans la fonction remove :

```

static int foo_remove(struct platform_device *pdev)
{
    struct foo_device *foodev;
    foodev = platform_get_drvdata(pdev);
    misc_deregister(&foodev->miscdev);
    /* ... */
}

```

2. Un lien entre l'instance de la structure struct miscdevice et l'instance struct device (champ dev de la structure platform_device) (voir (1)).

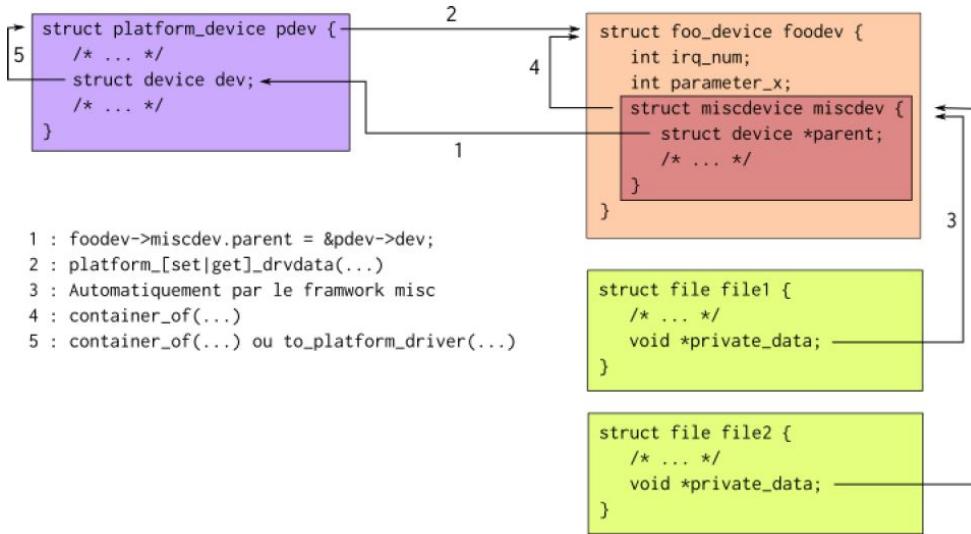


Figure 34: Schéma des liens entre ces différentes structures

Le framework misc met en place un troisième lien automatiquement entre l'instance de la structure struct file passée aux fonctions open, read, write... et l'instance correspondante de la structure struct miscdevice via le champ private_data de struct file. Exemple :

```
int foo_open(struct inode *inode, struct file *file)
{
    struct miscdevice *mdev = file->private_data;
    /* ... */
}
```

Finalement, il permet la communication entre les deux côté via des interface ‘open’, ‘read’, ‘write’...

5.2 Driver de PS2PL et PL2PS

Quand les parties fonctionnent correctement, il faut les deux parties communiquer pour échanger les informations. Par exemple, quand le résultat de traitement d'image indique que la voiture doit tourner à gauche, la partie d'ARM va écrire dans les registres du module ‘PS2PL’ créé par Artur. D'ailleurs, si le ultrason détecte un obstacle très proche, il va écrire dans le module ‘PL2PS’ et ce comportement peut générer une interruption par module ‘PL2PS’ pour annoncer la partie d'ARM. Au-dessous est le device tree de ces deux modules.

```
instruction_PL2PS_PS2PL_0: instruction_PL2PS_PS2PL@43c10000 {
    compatible = "xlnx,instruction-PL2PS-PS2PL-1.0";
```

```

    reg = <0x43c10000 0x10000>;
    xlnx,s00-axi-addr-width = <0x5>;
    xlnx,s00-axi-data-width = <0x20>;
};

instruction_PL2PS_PS2PL_1: instruction_PL2PS_PS2PL@43020000 {
    compatible = "xlnx,instruction-PL2PS-PS2PL-1.0";
    interrupt-parent = <&ps7_scugic_0>;
    interrupts = <0 31 1>;
    reg = <0x43020000 0x10000>;
    xlnx,s00-axi-addr-width = <0x5>;
    xlnx,s00-axi-data-width = <0x20>;
};

};

Pour enregistrer interruption dans Linux du module ‘PL2PS’, nous avons ajouté le numéro d’interruption et son type d’interruption est ‘rising edge’, c'est-à-dire juste un cycle d’horloge.

Ensuite, nous utilisons le même ‘misc framework’, et le résultat est montré dans la figure ?

```

The screenshot shows a terminal window titled 'Terminal' with the following content:

```

root@zynq:/dev# ls
block      lloop0      mmcblk0      port      ram15      snd      tty14      tty24      tty34      tty44      tty54      tty7      vcs6      xdevcfg
bus       lloop1      mmcblk0p1    psaux     ram2      stderr    tty15      tty25      tty35      tty45      tty55      tty8      vcs7
char      lloop2      mmcblk0p2    ptmx      ram3      stdin     tty16      tty26      tty36      tty46      tty56      tty9      vcsa
cpu_dma_latency lloop3      my_pl2ps   pts       ram4      stdout    tty17      tty27      tty37      tty47      tty57      ttyPS0  vcsal
disk      lloop4      my_ps2pl   ram0       ram5      tty      tty18      tty28      tty38      tty48      tty58      urandom vcsa2
fd        lloop5      my_xvdma_filter ram1      ram6       tty0      tty19      tty29      tty39      tty49      tty59      vcs      vcsa3
full      lloop6      my_xvdma_original ram10     ram7       ram1      ram8      tty10      tty20      tty30      tty40      tty50      tty60      vcs1      vcsa4
iio:device0 lloop7      network_latency ram11     ram9       ram12     ram10      tty11      tty21      tty31      tty41      tty51      tty61      vcs3      vcsa5
input      mem        null        ram13     random    ram12      tty12      tty22      tty32      tty42      tty52      tty62      vcs4      vcsa6
kmsg      memory_bandwidth ozwpan   ram14     shm       tty13      tty23      tty33      tty43      tty53      tty63      vcs5      vga_arbiter
root@zynq:/dev# lsmod
Module           Size  Used by
mysp2pl          2810  0
xilinx_vdma     11262  0
root@zynq:/dev# cat /proc/interrupts
CPU0            CPU1
27:             0      0      GIC  27  gt
29:  149139  112415  GIC  29  twd
35:             0      0      GIC  35  f800c000.ps7-ocmc
39:             43     0      GIC  39  f8007100.ps7-xadc
40:             0      0      GIC  40  f8007000.ps7-dev-cfg
43:             0      0      GIC  43  ttc_clockevent
45:             0      0      GIC  45  f8003000.ps7-dma
46:             0      0      GIC  46  f8003000.ps7-dma
47:             0      0      GIC  47  f8003000.ps7-dma
48:             0      0      GIC  48  f8003000.ps7-dma
49:             0      0      GIC  49  f8003000.ps7-dma
51:             0      0      GIC  51  e000d000.ps7-qspi
53:             0      0      GIC  53  ehci_hcd:usb1
54:  2391442     0      0      GIC  54  eth0
56:  24754      0      0      GIC  56  mmc0
61:  84107      0      0      GIC  61  xilinx-vdma-controller
62:  84042      0      0      GIC  62  xilinx-vdma-controller
63:             0      0      GIC  63  mysp2pl-controller
72:             0      0      GIC  72  f8003000.ps7-dma
73:             0      0      GIC  73  f8003000.ps7-dma

```

Figure 35: Résultat d’interruption et de périphérique

6. Traitement d’image

6.1. Introduction des outils

6.1.1 OpenCV

Pour traiter des images originaires du camera, la librairie OpenCV (pour Open Computer Vision) est choisie, car c'est une bibliothèque qui met à disposition de nombreuses

fonctionnalités très diversifiées permettant de créer des programmes partant des données brutes pour aller jusqu'à la création d'interfaces graphiques basiques.

Néanmoins, la performance de traitement d'images est souvent limitée par la taille de mémoire et la fréquence de lecture et d'écriture liée au mémoire.

6.1.2 Vivado HLS

La synthèse de haut niveau (HLS), parfois appelée la synthèse de C, la synthèse algorithmique, ou la synthèse du comportement, est un processus de conception automatisé qui interprète une description algorithmique d'un comportement désiré et crée du matériel numérique mettant en œuvre ce comportement. La synthèse commence par une spécification de haut niveau du problème, où le comportement est généralement découpé de par exemple synchronisation au niveau de l'horloge. Début HLS explore une variété de spécifications d'entrée langues, bien que les applications de recherche et commerciaux récents généralement reconnus sous-ensembles synthétisables de ANSI C / C ++ / SystemC / Matlab. Le code est analysé, architecturalement contraint, et prévu de créer un niveau registre de transfert (RTL) langage de description matérielle (HDL), qui est à son tour généralement synthétisés au niveau de la porte par l'utilisation d'un outil de synthèse logique. Le but de HLS est de laisser les concepteurs de matériel efficacement construire et vérifier le matériel, en leur donnant un meilleur contrôle sur l'optimisation de leur architecture de conception, et par la nature de permettre au concepteur de décrire la conception à un niveau d'abstraction plus élevé tandis que l'outil fait la mise en œuvre RTL. Vérification de la RTL est une partie importante du processus.

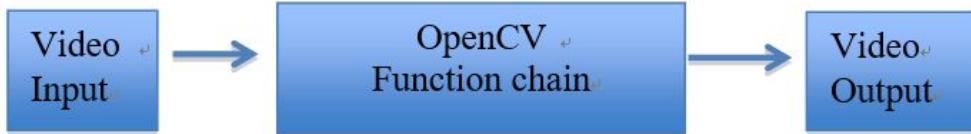
Vivado HLS est un outil développé par l'entreprise de Xilinx pour réaliser la synthèse de haut niveau et finalement générer IPs qui peuvent être intégrés dans la partie de FPGA.

Normalement, il y a quatre étapes nécessaires de générer IPs en utilisant Vivado HLS, ils sont la C simulation, C synthèse, C-RTL co-simulation et l'exportation de IP. Dans le répertoire de IPs, il y a un sous-répertoire que s'appelle driver, c'est un répertoire contenant des fichiers ou des fonctions de l'initialisation, de l'interruption et du démarrage sont définies, pour faire bien fonctionner des IPs en FPGA.

La simulation de C est un processus de présynthèse pour vérifier la conception, elle permet une vérification rapide et libre sur n'importe quel système et d'assurer la rectitude de l'algorithme avant de la génération de RTL.

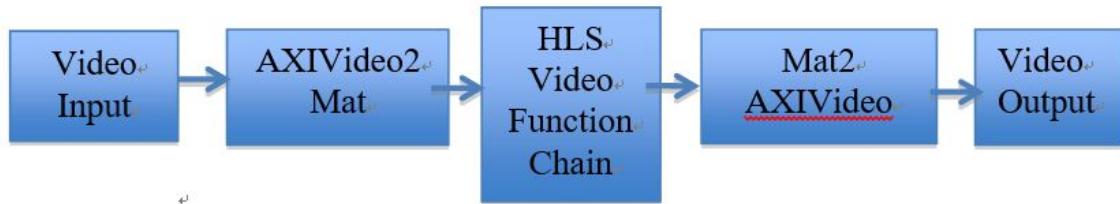
La C-RTL co-simulation est un processus de post-synthèse pour vérifier la conception de RTL en fonction du testbench original.

Pour réaliser la C synthèse, il faut mettre des descriptions nécessaires par exemple l'implémentation de l'interface AXI-Lite pour le port de retour et les paramètres, mettre des descriptions optionnelles afin de optimiser du temps, de l'aire et de performance, par exemple l'implémentation de BRAM aux tableaux, l'implémentation de Pipeline aux boucles, l'implémentation de Dataflow parmi des fonctions, et l'utilisation des types de données avec le bitwidth réduit(ex: ap_int< >, ap_uint< >, ap_fixed< > en C++). Quand le traitement de vidéo est réalisé par l'application de OpenCV, la chaîne est comme la graphique dessous.



Toutefois la performance de OpenCV est limitée par les paramètres liés au mémoire, Vivado peut être utilisé pour augmenter la performance de traitement, car la fréquence de traitement en FPGA basée sur le pixel atteint milliard fois par seconde, mais la fréquence de traitement en ARM basée sur la frame est million fois par second, on peut donc combiner le traitement en FPGA et en ARM dédiant les traitements différents.

Quand le traitement est accéléré en utilisant des modules synthétisables en RTL, le schéma devient :



De plus, HLS Vidéo bibliothèque est un C / C ++ bibliothèque fournie avec Vivado HLS pour aider à accélérer la vision par ordinateur/ applications de traitement d'image sur FPGA. Il comprend couramment utilisé des structures de données, des interfaces OpenCV, AXI4-Stream I / O, et des fonctions de traitement vidéo. HLS Vidéo bibliothèque utilise des bibliothèques OpenCV comme modèle de référence, la plupart des fonctions de traitement vidéo à l'interface similaire et le comportement équivalent avec des fonctions de OpenCV correspondantes. Les bibliothèques prédéfinies OpenCV sont également livrées avec Vivado HLS sur différentes plates-formes, afin que les utilisateurs soient en mesure d'utiliser OpenCV directement, sans effort supplémentaire.

Dans un système typique de vidéo en utilisant des fonctions d'OpenCV, la majeure partie de l'algorithme restera sur le processeur. Seules les parties de l'algorithme qui nécessitent l'accélération dans le tissu de FPGA seront synthétisées et donc mis à jour pour utiliser la bibliothèque vidéo HLS. Les fonctions d'interface OpenCV sont prévues pour assurer le transfert de données entre le code d'OpenCV exécuté sur le processeur et la fonction du matériel synthétisé en cours d'exécution sur le tissu FPGA. En plus, les fonctions comprennent également des moyens de conversion de formats de données OpenCV vers et depuis les types de données HLS Vidéo bibliothèque, par exemple `hls :: Mat`. Pour utiliser les fonctions de l'interface OpenCV, le fichier d'en-tête `hl_opencv.h` doit être inclus. Ces fonctions pourraient être utilisées dans le code qui reste sur le processeur.

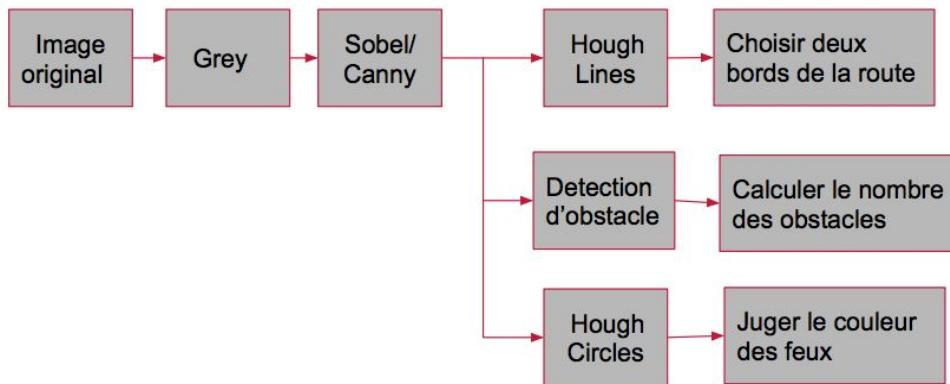
La plupart des fonctions de bibliothèque vidéo pour accélérer le traitement vidéo est d'utiliser `hls :: Mat` que le type d'une image de données. Dans le composant synthétisable de traitement vidéo, les fonctions d'E / S qui convertissent `hls :: Mat` de / vers AXI4-Stream type de données compatible (`hls:: flux`) sont fournies. Pour utiliser les fonctions AXI4-Stream I / O, le fichier d'en-tête `hl_video.h` doit être inclus. Ces fonctions d'E / S doivent être invoquées à l'intérieur de la fonction du matériel, et elles sont synthétisables.

Les fonctions de traitement vidéo fournis dans la bibliothèque vidéo HLS sont spécifiquement pour manipuler des images vidéo. La plupart de ces fonctions sont conçus pour accélérer les fonctions de OpenCV correspondants, qui ont la signature et l'utilisation similaire.

6.2 Traitement en logiciel

Pour bien choisir la direction des mouvements de la voiture, trois critères sont considérés, qui sont séparément la direction déterminée par les bords de route, la direction déterminée par des obstacles détectés, et le couleur des feux de circulation.

Le schéma de traitement en logiciel est conçu comme la graphique dessous:



En fonction de trois branches de traitement, les trois directions bases sur de différents critères de mouvement de voiture seront obtenues. Après les avoir obtenu, la direction finale sera décidée par certaine priorité.

Puisque on n'a pas de scène réelle à tester, on utilise la scène de cuisine pour la simulation en donnant qu'il y a des lignes sur le sol, et on peut mettre des bouteilles comme obstacles.

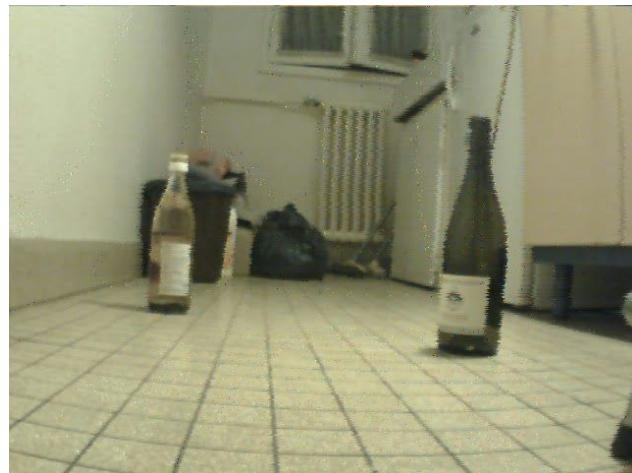
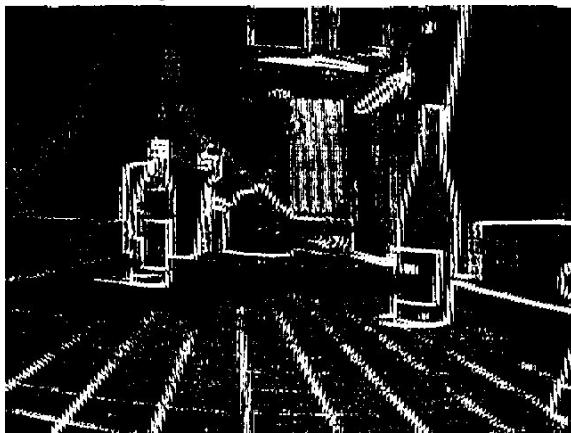


Image original

6.2.1 Grey, Sobel/Canny

Tout a bord, il faut extraire des informations de contours, la conversion d'une image RGB à une image gray est premièrement nécessaire. Normalement, le gradient est utilisé pour détecter des contours. Sobel et Canny sont deux fonctions plus courantes à détecter des

contours, Sobel est plus simple et Canny a un meilleur résultat car il peut supprimer plus de points dérangeants. C'est un exemple des traitements par Sobel et Canny.



Sobel



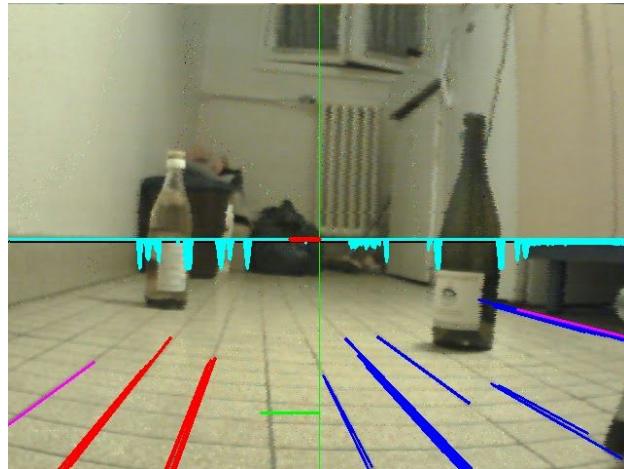
Canny

6.2.2 HoughLines

Après avoir des contours, le premier traitement est de détecter deux bords de la route. Dans la scène en réalité, on voit que la route apparaît toujours dans la moitié d'image basse, donc on traite la moitié basse d'image pour détecter la route. En utilisant la méthode de HoughLines fournie par OpenCV, certaines lignes peuvent être détectées comme indiqués dans les images dessous. Pour simplifier la sélection, des lignes avec du gradient trop petit ou trop grand sont supprimées en premier. Des lignes restes sont classées par des lignes gauches et des lignes droites en fonction de pente. Parmi des lignes gauches, les pentes sont négatives, si on veut choisir la ligne plus loin du centre, la ligne avec la pente plus grande sera choisie. Pour la côte droite, c'est la ligne avec la pente plus petite qui est choisie.

Etant donné des deux bords, la direction peut être déterminée par un algorithme adapté. Ici on peut calculer le point croisé par les deux lignes, si l'abscisse du point est inférieur de la moitié de largeur d'image, la direction de gauche est donnée, sinon la direction de droite est donnée. Cependant, pour que la voiture ne change pas toujours la direction, on peut mettre un critère où si la distance du point à la ligne au milieu de l'image ne dépasse pas un nombre de pixels, on peut maintenir la direction donnée avant.

Dans l'image dessous, des lignes gauches sont marquées par la couleur rouge, des lignes droites sont marquées par la couleur bleue, des deux lignes de la route sont marquées par la couleur pourpre, et la petite ligne verte est la direction.



6.2.3 Détection d'obstacle

Deuxièmement, on fait la détection des obstacles. Sur l'image de Sobel, tous les pixels marqués comme contours sont blanc, des obstacles donc peuvent être détectés par compter le nombre des pixels blanc.

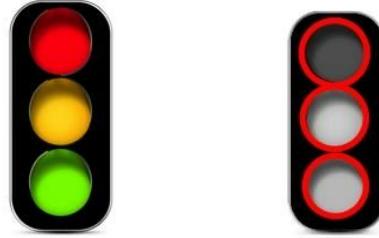
Encore pour la simplification des calculs, la moitié d'image de milieu est choisi à détecter des obstacles. La première étape est de trouver une ligne presque au milieu de l'image ou des obstacles se trouvent autour de cette ligne, pour le faire, on compte des trois lignes avec plus de pixels blanc et choisit juste une qui a la variance plus petite. Ensuite on peut compter des pixels blancs au-dessous de la ligne avec une distance par colonne, des colonnes avec des pixels plus nombreux que des pixels en moyenne de cette ligne sont dessinés avec la longueur correspondante le nombre de pixels blanc. En même temps, des pixels blancs sont accumulés séparément dans trois parties, la partie de gauche, la partie de droit et la partie au milieu, si la moyenne de pixels blanc dans la partie du milieu est plus grande que la moyenne globale de la ligne, on peut dire que la voiture ne doit pas avancer, notamment elle faut soit tourner à gauche, soit tourner à droite en comparant des nombres de pixels blanc dans la partie de gauche et la partie de droit. Sinon la voiture peut rouler tout droit.

Dans l'image de l'étape précédente, la ligne d'obstacle est bleue avec des petites lignes verticales, et la petite ligne rouge sur la ligne d'obstacle indique la direction calculée selon des obstacles.

6.2.4 HoughCircles

Dans la route en réalité, le feu de circulation est un élément important, pour bien simuler la scène, on fait aussi la détection des feux. Comme les feux de circulation apparaissent dans le haut de la scène, on choisit la partie haute pour détecter des feux.

En utilisant la fonction HoughCircles fournie par OpenCV, des cercles ayant le rayon limité par des deux dernières paramètres de cette fonction sont détectés avec la coordonnée de centre et le rayon.



Alors des régions des cercles sont extraites sur l'image originale et marque comme des régions intéressantes car des informations nécessaires sont déjà obtenues dans l'étape précédente. En fonction des informations de couleur de l'image originale, on peut différencier que le feu est rouge, jaune, verre, ou éteinte. Car le couleur jaune est compose par le couleur rouge et le couleur verre, pour chaque région intéressante, on additionner des valeurs des pixels par channel rouge, jaune et vert, ensuite calcule ratios entre ces valeurs.

Au-dessous est le code principale de juger le couleur de la région intéressante. S'il y a du feu rouge ou jaune parmi des cercles détectés, on la donne la commande d'arrêter, sinon on la donne la commande de continuer.

```

for (int i = 0;i<HEIGHT;i++)
{
    for (int j = 0;j<WIDTH;j++)
    {
        rData += Rmat.at<unsigned char>(i,j);
        gData += Gmat.at<unsigned char>(i,j);
        bData += Bmat.at<unsigned char>(i,j);
    }
}

rData = rData/(HEIGHT * WIDTH);
gData = gData/(HEIGHT * WIDTH);
bData = bData/(HEIGHT * WIDTH);

rij = rData /255;
gij = gData /255;
bij = bData /255;

rg=rData/gData;
rb=rData/bData;

gr=gData/rData;
gb=gData/bData;

if(rg>=2 && rb >=2 && gr <= 1) { //red
    return 1;
}
else if(gr>=2 && gb >=2 && rg <= 1) { //green
    return 2;
}
else if(rij <= 0.3 && gj <= 0.3 && bij <= 0.3){ //off
    return 0;
}
else{ //yellow
    return 3;
}

```

6.2.5 Stratégie de prendre la décision

Ayant trois directions bases sur la détection de route, d'obstacle et de feu, il faut décider laquelle a une priorité.

En considérant la situation réelle et la précision des algorithmes, on propose que la détection de feu ait la priorité plus haute, suivi par la détection de route, et la détection d'obstacle. C'est à dire que si le résultat de détecter le feu est la continuation, on prend en

compte des deux autres décisions, sinon on l'arrête directement. Si la voiture peut continuer, et la direction donne par le résultat de détection de route est l'avancement tout droit, le résultat final est décidé par la détection d'obstacle, si la direction base sur la détection de route est de tourner à gauche ou à droite, c'est alors le résultat final.

6.3 Traitement en matériel

Même si des fonctions sont toutes réalisées en logiciel, pour ce cas, un objectif critique est que le traitement faut être temps réel. Le traitement en logiciel ne satisfait pas suffisamment cet objectif, donc pour l'accélération du traitement on peut mettre plusieurs sous-traitements en matériel en utilisant FPGA car le traitement par pixel sur FPGA peut arriver milliard fois par seconde.

Pour bien savoir l'optimisation en utilisant l'outil Vivado HLS, il faut d'abord bien comprendre des algorithmes relatifs.

6.3.1 Sobel

En théorie, des calculs de Sobel sont indiqués dans les équations au-dessous:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Où A est un morceau d'image gray, représenté par un matrix 3*3.

On voit qu'il existe le calcul radical qui peut être simplifié par ce calcul.

$$|G| = |G_x| + |G_y|$$

Bien que c'est un traitement sur pixel, il est nécessaire pour obtenir le résultat d'un pixel de stocker un matrix de 3*3 pixels gray qui sont des voisins de ce pixel. Des types de hls::Buffer et hls::Window sont utiles pour ce traitement pour stocker des pixels nécessaires au long de la sortie du stream de pixels. Et des fonctions liées comme shift_right(), shift_down() et insert() sont fréquemment utilisées pour le déplacement.



C'est le IP de Sobel généré par Vivado HLS qui est intégré dans le système final.

6.3.2 Canny

En théorie, le Canny est calculé comme des étapes dessous:

La première étape est de réduire le bruit de l'image originale avant d'en détecter les

contours. Ceci permet d'éliminer les pixels isolés qui pourraient induire de fortes réponses lors du calcul du gradient, conduisant ainsi à de faux positifs. La fonction de GaussianBlur fournie par la bibliothèque hls_video peut simplement être appelée ici.

Après le filtrage, l'étape suivante est d'appliquer un gradient qui retourne l'intensité des contours. L'opérateur utilisé permet de calculer le gradient suivant les directions X et Y, il est composé de deux masques de convolution, un de dimension 3×1 et l'autre 1×3 :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad ; \quad G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

La valeur du gradient en un point est approximée par la formule comme dans la méthode dans Sobel:

$$|G| = |G_x| + |G_y|$$

Les orientations des contours sont déterminées par la formule :

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Nous obtenons finalement une carte des gradients d'intensité en chaque point de l'image accompagnée des directions des contours. Mais le calcul de fonction trigonométrique est compliqué à appliquer en utilisant FPGA, donc il peut être approximé par ce calcul:

$$\Theta = \text{PI}/4 * (1 - (G_x - G_y)/(G_x + G_y))$$

Finalement la dernière étape est la suppression du non-maxima, La carte des gradients obtenue précédemment fournit une intensité en chaque point de l'image. Une forte intensité indique une forte probabilité de présence d'un contour. Toutefois, cette intensité ne suffit pas à décider si un point correspond à un contour ou non. Seuls les points correspondant à des maxima locaux sont considérés comme correspondant à des contours, et sont conservés pour la prochaine étape de la détection. Un maximum local est présent sur les extrêmes du gradient, c'est-à-dire là où sa dérivée s'annule.

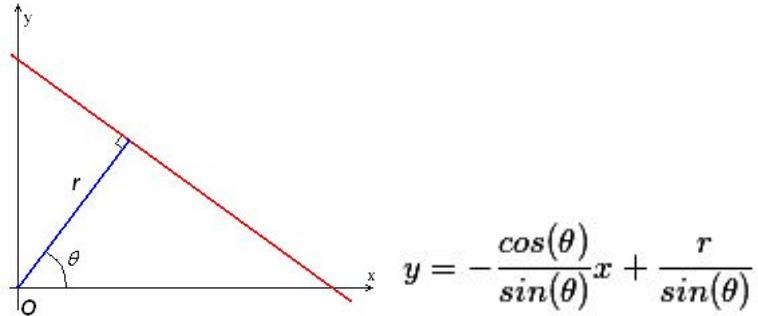
C'est une étape importante pourquoi on voit que le contour de Canny est plus fin que le contour de Sobel dans des images mise précédentes. Dans l'application de cette étape, des comparateurs dans le FPGA sont utilisés.

6.3.3 HoughLines

Ayant identifié tous les points de contours de cette image à l'aide de Sobel ou Canny, au point (x, y) du contour, on fait donc correspondre une courbe $[\theta, p]$, où θ prend toutes les valeurs possibles de 0 à 2π , p prenant la valeur $p = x * \cos(\theta) + y * \sin(\theta)$. Une fois appliquée à tous les points des contours, et, pour chacun, à tous les angles θ possibles, les couples (θ, p) les plus souvent adressés sont les coordonnées des droites ou des segments de droite les plus représentés dans l'image de départ.

Ensuite on peut choisir certain nombre de lignes selon le paramètre de MaxLines qui sont

les plus souvent votés.



Dans cette méthode, il est évident que l'on a besoin de nombreux de registres et accumulateurs pour compter et stocker des votes aux lignes différentes.

Pour réduire des ressources nécessaires, on peut faire plusieurs optimisations selon des paramètres de fonction. En analysant le calcul de HoughLines, on voit que les résolutions de rho et thêta, aussi le nombre de lignes sont des paramètres importantes car ils peuvent déterminer le nombre de boucles. Par conséquent, on a des optimisations possibles qui sont de limiter la variation de thêta puisque des angles de la route du caméra sont impossible d'être trop petite ou trop grand, de augmenter des résolutions des rho et thêta, et de réduire le nombre de lignes demandées. Mais c'est sûr que l'on va perdre certaine informations. A propos du calcul de fonction trigonométrique, on peut pré-calculer des résultats et les stocker dans un tableau parce que c'est plus facile de consulter des tableaux que de calculer des valeurs directement.

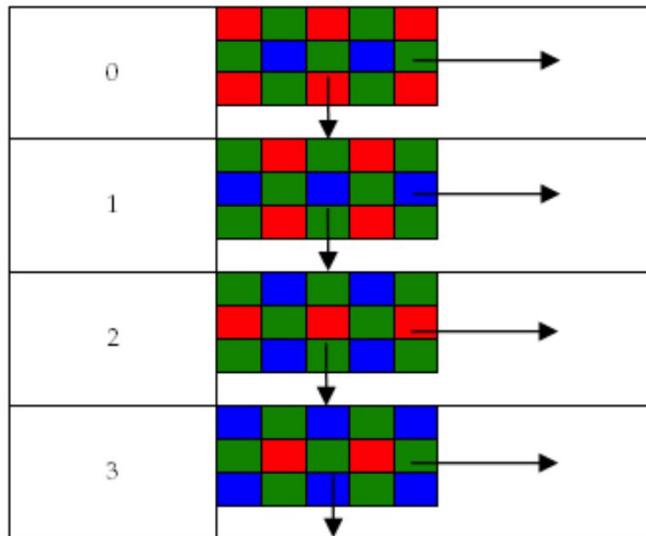
En ce qui concerne les résultats de HoughLines, on a deux façons de les montrer. On peut choisir de sortir un stream d'image avec les lignes détectées, dans ce cas, il faut encore plus de boucles pour confirmer si un pixel est sur une ligne désirée ou pas. Une autre façon est de calculer les couples (θ, ρ) de paramètres des lignes souhaitées et passer les résultats à la partie de system, et enfin on montre les lignes dans les traitements suivants en logiciel. De cette façon, des ressources requis sont encore réduites.

6.3.4 CFA

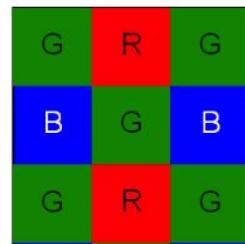
Sauf des méthodes indiquées avant, un autre module est indisponible car l'image sortant du camera est Bayer. Pour que des traitements fonctionnent bien, il faut convertir l'image en format de Bayer à l'image en format de RGB.

Xilinx fournit un IP de CFA à appliquer cette conversion, mais il est un IP payant. En analysant l'approche, on génère un IP similaire.

Bayer a 4 phases différents possibles qui sont indiqués dans la graphique suivante.



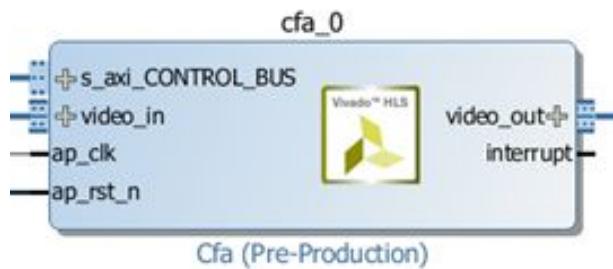
En prenant la deuxième phase comme exemple, on fait la conversion avec l'interpolation.



$$\begin{aligned} \text{Interpolation: } R &= (R_{0,1} + R_{2,1})/2 \\ G &= (G_{0,0} + G_{0,2} + G_{2,0} + G_{2,2})/4 \\ B &= (B_{1,0} + B_{1,2})/2 \end{aligned}$$

Selon l'interpolation, il y a pas huit voisins pour un pixel au bord d'image. Dans notre cas, on a décidé de négliger ces pixels car ils ont peu d'influence sur la détection des contours même si il y a des bordures noires dans l'image convertie.

L'IP de CFA crée par nous-même apparaît comme cette graphique.



6.4 Résultats

6.4.1 Utilisation et performance

Pour ces modules réalisés en utilisant Vivado HLS, on peut avoir des rapports sur la performance et l'utilisation des ressources sur la carte.

```

=====
== Vivado HLS Report for 'sobel'
=====
* Date:      Mon Jan 25 00:02:04 2016
* Version:   2015.2 (Build 1266856 on Fri Jun 26 16:57:37 PM 2015)
* Project:   sobel_n
* Solution:  solution1
* Product family: zynq
* Target device: xc7z010clg400-1

=====
== Performance Estimates
=====
+ Timing (ns):
 * Summary:
 +-----+
 | Clock | Target| Estimated| Uncertainty|
 +-----+
 | ap_clk | 40.00| 35.80| 5.00|
 +-----+

+ Latency (clock cycles):
 * Summary:
 +-----+
 | Latency | Interval | Pipeline |
 | min | max | min | max | Type |
 +-----+
 | 7| 2083090| 4| 2083089| dataflow |
 +-----+

```

```

=====
== Utilization Estimates
=====
* Summary:
+-----+
| Name | BRAM_18K| DSP48E| FF | LUT |
+-----+
| Expression | -| -| 0| 2|
| FIFO | 0| -| 150| 1099|
| Instance | 3| 19| 1810| 2390|
| Memory | -| -| -| -|
| Multiplexer | -| -| -| 10|
| Register | -| -| 12| -|
+-----+
| Total | 3| 19| 1972| 3501|
+-----+
| Available | 120| 80| 35200| 17600|
+-----+
| Utilization (%) | 2| 23| 5| 19|
+-----+

```

En regardant cet exemple, on peut bien savoir que le temps et la latence satisfaisant bien les exigences, l'utilisation des ressources n'est pas nombreuse. Mais pour des modules comme Canny et HoughLines, l'utilisation des ressources sont presque pleines qu'on peut voir dans le rapport au-dessous.

```

=====
= Vivado HLS Report for 'canny'
=====
* Date:      Sun Feb 07 15:03:48 2016
* Version:   2015.2 (Build 1266856 on Fri Jun 26 16:57:37 PM 2015)
* Project:   canny
* Solution:  solution1
* Product family: zynq
* Target device: xc7z010clg400-1

=====
= Performance Estimates
=====
+ Timing (ns):
 * Summary:
 +-----+
 | Clock | Target| Estimated| Uncertainty|
 +-----+
 | ap_clk | 40.00| 34.71| 5.00|
 +-----+

+ Latency (clock cycles):
 * Summary:
 +-----+
 | Latency | Interval | Pipeline |
 | min | max | min | max | Type |
 +-----+
 | 279 | 5859091| 278| 5859063| dataflow|
 +-----+

```



```

=====
= Utilization Estimates
=====
* Summary:
 +-----+
 | Name      | BRAM_18K| DSP48E| FF    | LUT    |
 +-----+
 | Expression |      -|      -|     0|      7|
 | FIFO      |      0|      -| 207| 1047|
 | Instance   |     12|     58| 6269| 15337|
 | Memory    |      0|      -|   48|      3|
 | Multiplexer|      -|      -|     -|     30|
 | Register   |      -|      -|   32|      -|
 +-----+
 | Total      |     12|     58| 6556| 16424|
 +-----+
 | Available  |    120|     80| 35200| 17600|
 +-----+
 | Utilization (%) | 10| 72| 18| 93|
 +-----+

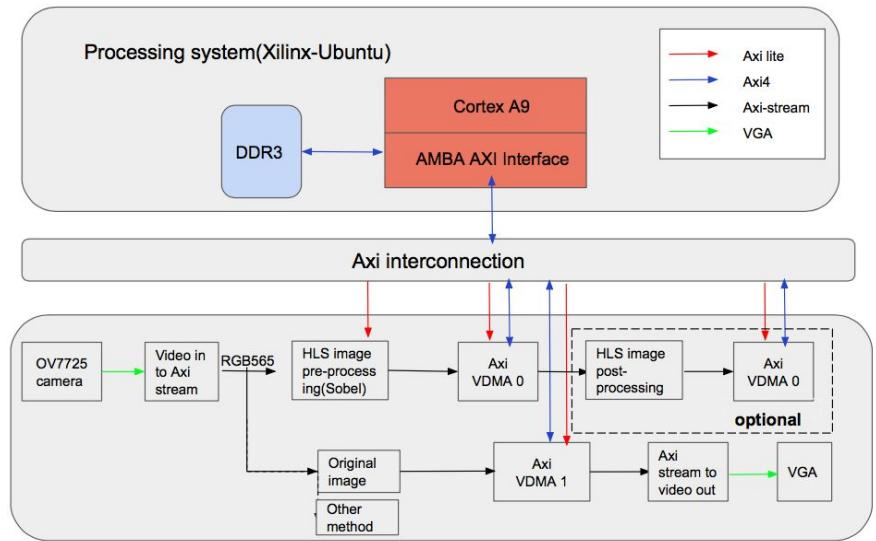
```

Si ces modules sont intégrés dans le système matériel, il y aura plus de ressources pour d'autres modules nécessaires. Par conséquent, finalement ces modules ne sont pas intégrés dans la partie FPGA.

Pour l'amélioration future, la substitution du calcul de division peut réduire l'utilisation des ressources.

6.4.2 Schéma final

Après avoir vu tous les performances et les utilisations des modules, finalement on a choisi le traitement de CFA et Sobel d'être réalisé en matériel, les traitements restes sont réalisés en logiciel.



6.4.3 Comparaison du temps

Pour des images 640*480 sortant du camera, si elles sont traites en OpenCV sur la carte Zybo, les temps de sous-traitements sont comptes et indiques dans ce tableau:

Fonction	Sobel	Canny	HoughLines
Temps	0.07-0.1s	0.1-0.15s	0.2-0.3s

Quand ces méthodes sont appliques en matériel, les temps sont beaucoup réduites, dans notre cas, la fréquence de système est 25M, alors 60 frames peuvent être traites par second, c'est à dire que toutes ces méthodes peut finir dans 0.017s.

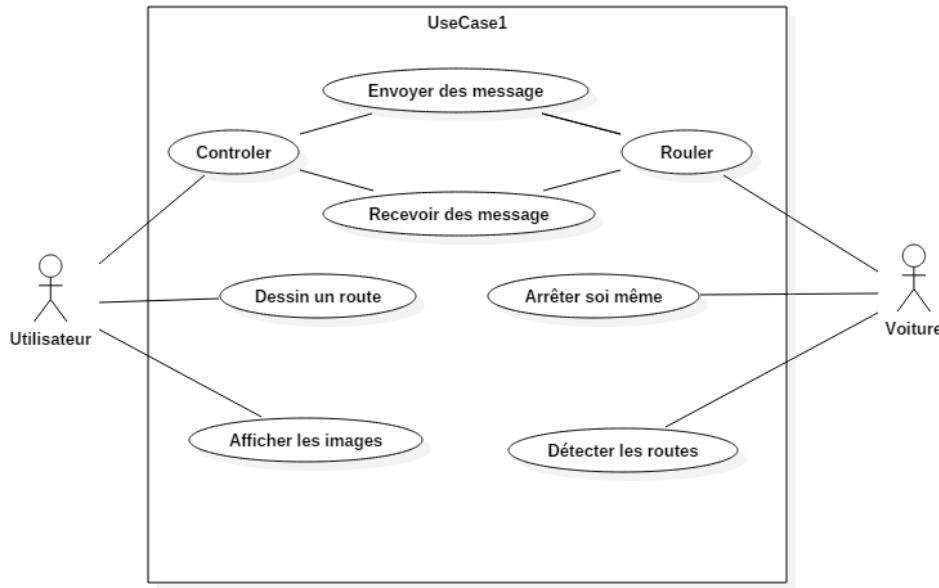
7 Application portable

On voudrait utiliser l'application mobile pour contrôler la petite voiture.

On choisit la technologie <Bluetooth> pour la communication entre la voiture et l'application. On peut également transformer l'image que la voiture capture en utilisant le protocole TCP/IP. Et puis, si cela marche bien, on voudrait qu'on puisse transformer les informations en WIFI.

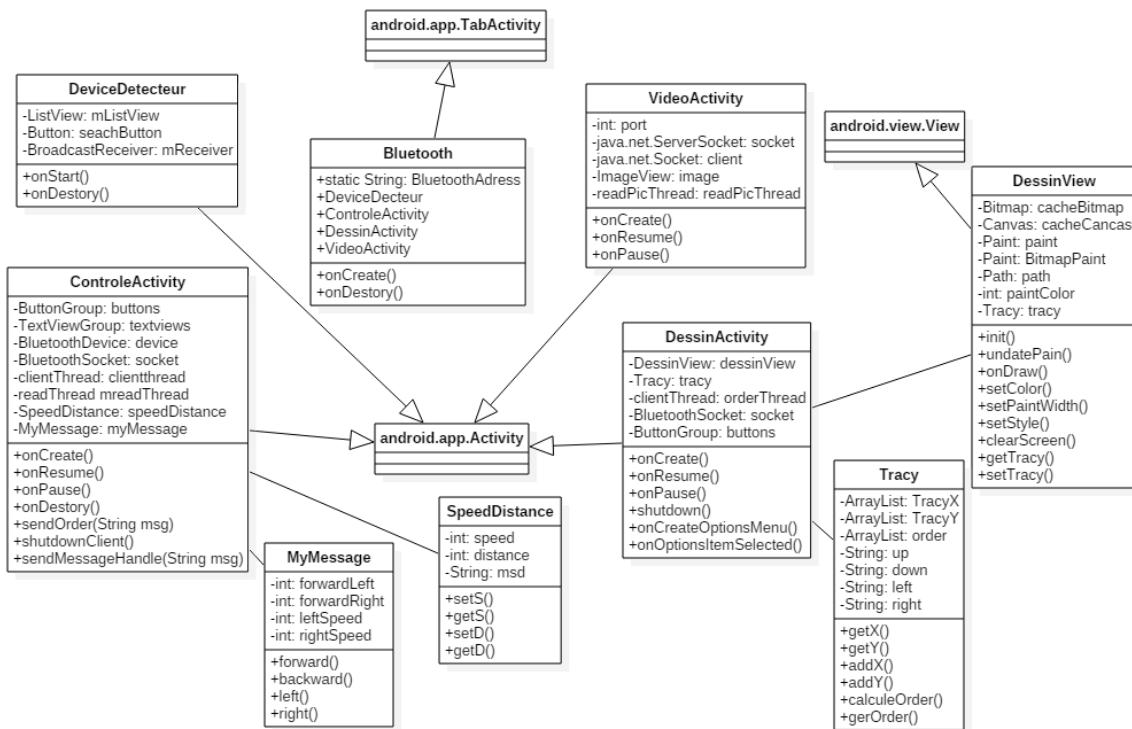
7.1 Architecture

7.1.1 Cas d'utilisation



Dans notre cas, il y a deux acteurs principaux, l'utilisateur et la voiture. Pour l'utilisateur, on voudrait que il puisse contrôler la voiture, dessin un route et voir les images que la voiture capture. Il peut aussi envoyer des messages à la voiture et recevoir des messages de voiture comme la vitesse et la distance. Pour la voiture, on voudrait qu'elle puisse rouler selon les messages que l'on lui envoie, et aussi envoyer des messages comme la vitesse, la distance. La voiture peut aussi s'arrêter soi même quand elle détecte que il y a l'obstacle. Elle peut détecter les routes parce qu'elle a une caméra, on va développer un algorithme pour analyser les images qu'elle capture. Les deux acteurs vont se communiquer selon le Bluetooth et aussi le protocole TCP/IP.

7.1.2 Diagramme de classe



Dans notre cas, pour bien communiquer entre l'utilisateur et la voiture, on a besoin neuf classes ensemble. On va développer cette application en Java, donc on va utiliser le fichier XML pour afficher les vues au niveau View, en utilisant le design pattern MVC. On a quatre sous classes comme les contrôles au niveau Control, ils sont DeviceActivity, ControlActivity, DessinActivity et VideoActivity. Ils tous héritent la classe android.app.Activity. On a des classes MyMessage, SpeedDistance et Tracy pour stocker des données, au niveau Model.

7.1.3 Communication avec la voiture

Pour bien contrôler la voiture, la partie matérielle et la partie logiciel doivent bien collaborer. On a décidé comme peut-on échanger la message, par exemple, pour avancer la voiture, il faut donner une signal comme « W0055 », « W » signe que c'est un message pour contrôler la moteur de la voiture, « 00 » signe que la côté gauche et la côté droite est dans la direction d'avant (par contre 1 signe que c'est la direction d'arrière), « «55 » signe que la vitesse est moyen, 0 est le minimum et 9 est la maximum. Si on envoie un message comme « S », la voiture va se retourner la vitesse de la voiture par un message commençant par « S ». Si on envoie un message comme « D », la voiture va se tourner la distance combien la voiture a déjà roulé. On peut aussi choisir la mode de contrôle, par exemple, si on envoie un message comme « M0 », c'est-à-dire que l'utilisateur contrôle la voiture complètement. Si on envoie « M1 », la voiture peut rouler soi-même, et elle va détecter des routes et des obstacles soi-même. On a développé cette partie même si la voiture ne marche pas vraiment bien.

C'est que l'on peut faire plus loin est de créer la troisième façon de contrôler. Par exemple on peut mélanger les deux premières façons. C'est-à-dire que on contrôle la voiture, mais quand elle détecte que il y a un obstacle, elle choisit une autre route un peu différent que l'on veut, mais arriver à la destination comme même. On croit que c'est une meilleure façon de contrôler même si on a pas bien développé.

7.2 Implémentation

Outil: Android Studio, Eclipse

On veut que l'on puisse contrôler la voiture en utilisant une console de jeux, dans ce cas, on peut développer une application Androïde. On choisit Android Studio pour développer le code, le code de contrôle est écrit en Java, l'interface utilisateur est en XML.



Figure 1 DeviceActivity

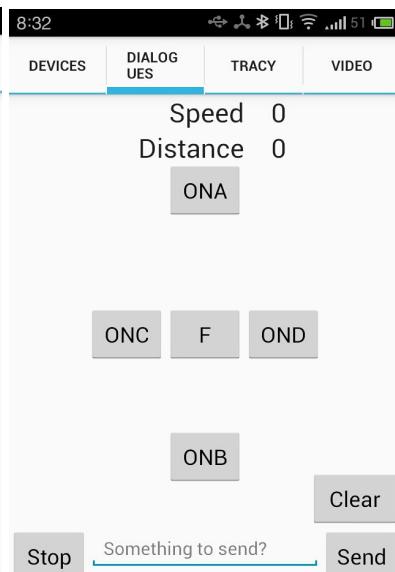


Figure 2 ConsoleActivity

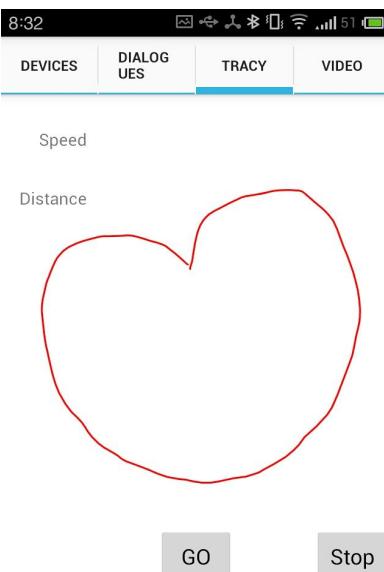


Figure 3 DessinActivity

Partie Java :

7.2.1 Classe Bluetooth.java

C'est une classe supérieure pour contrôler toutes les classes. C'est-à-dire que toutes les classes doivent inscrire à cette classe tout à bord et avoir un tag, et après ils peuvent afficher à l'écran.

7.2.1.1 DeviceActivity.java (Figure 1)

C'est une classe pour activer le bluetooth s'il n'est pas encore actif. Quand on active le bluetooth, on peut rechercher l'équipement que l'on peut utiliser, dans notre cas, c'est la petite voiture. Dès que l'on choisit un équipement et lui connecter, on saute à la classe ConsoleActivity.

7.2.2 Classe ConsoleActivity.java (Figure 2)

C'est notre classe principale pour contrôler la voiture. On a cinq boutons pour contrôler la voiture, « ONA » pour la direction d'avant, « ONB » pour la direction d'arrière, « ONC » pour la direction de gauche, « OND » pour la direction de droite et « F » pour terminer la voiture. Chaque bouton envoie les messages selon notre accord. De plus, on peut augmenter ou diminuer la vitesse si on appuie le bouton plusieurs fois.

7.2.2.1 MyMessage.java

C'est une classe supplémentaire de ConsoleActivity pour stocker les messages. Par exemple quand on appuie le bouton « ONA », le message devient « W0011 », si on appuie encore une fois, il devient « W0022 », et après « W0033 » etc... Il est désigné pour mémoire le statut de la voiture.

7.2.3 Classe DessinActivity.java (Figure 3)

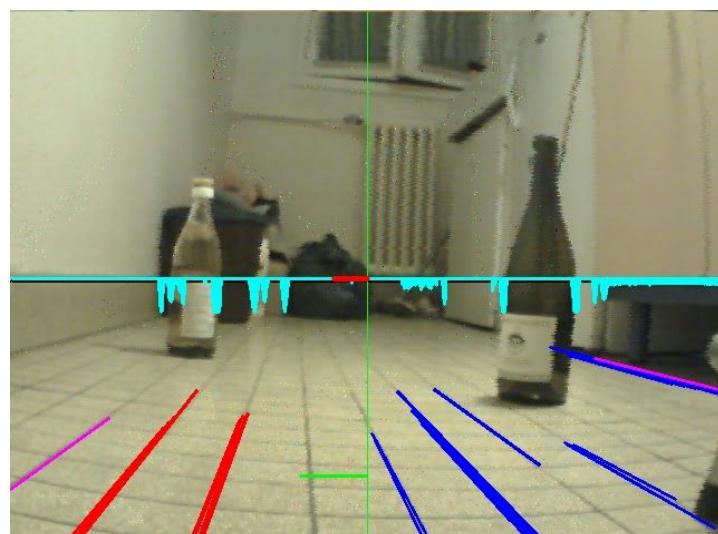
Cette classe est désigné pour s'amuser. C'est une fonctionnalité intéressante. Par exemple on dessin un route de cœur, la voiture va dessin un cœur sur terre !

7.2.3.1 Tracy.java

C'est la classe supplémentaire de DessinActivity pour stocker la route que l'on dessine. Quand on appuie sur l'écran, et après bouger sur l'écran, cette classe va stocker la route automatiquement. Quand on finit le dessin, et puis on appuie sur le bouton « GO », l'application va envoyer les messages à la voiture. Quand je développe le code la première fois, elle va juste envoyer tous les messages sans arrêt. Mais cela ne marche pas bien, donc je ajoute un intervalle de 200 milliseconde entre chaque message, il marche mieux.

7.2.4 Classe VideoActivity.java

C'est une classe pour afficher les images que la camera de la voiture capture. C'est-à-dire quand on ouvrir cet activité, on verra ce que la voiture voie. C'est une fonctionnalité utile. Par exemple si on ne peut pas rentrer dans une cave, la voiture peut nous aider. Il peut aussi afficher les images traités, comme figure suivant.



7.2.5 Classe SpeedDistance.java

C'est une classe désigné pour stocker la vitesse et la distance de la voiture. Quand on demande un « S » à la voiture, elle va se retour une vitesse, on utilise cette classe et après afficher la vitesse sur l'écran.

Partie XML :

7.2.6 consoleView.xml

On utilise le « RelativeLayout » pour afficher la vue. On a cinq boutons principaux.

7.2.7 tracy.xml

On utilise le « RelativeLayout » pour afficher la vue. On a un bouton et une plaque pour dessiner.

7.2.8 video.xml

On utilise le « LinearLayout » pour afficher la vidéo.

7.3 Challenge

Quand on développe cette application, comment traiter la concurrence entre les activités est un challenge. L’activité Console veut utiliser le socket pour transformer les informations. Mais l’activité Dessin veut aussi. Les deux activités ne peuvent pas utiliser le même socket parce qu’il y aura des concurrences. Pour traiter ce problème, on a deux solutions. La première qui est plus élégant est d’utiliser le service. C'est-à-dire on créer une autre classe supplémentaire « SocketService » pour gérer le socket. Quand une activité veut utiliser le socket, il demande au service. Le service va gérer le socket automatiquement. On a essayé l’utiliser mais cela ne marche pas bien. On ne trouve pas la solution même si on consulte la documentation d’Androïde de Google. On choisit une autre solution, gérer le socket soi-même. On a bien analyse les besoins et trouve que si une activité utilise le socket, une autre n’a pas besoin, donc on peut fermer le socket manuelle. C'est-à-dire quand on changer à une autre activité, on ferme le socket tout d’abord, et après ouvrir l’activité, et se connecter le socket. C'est une solution plus simple même si ce n'est pas facile à étendre.

On utilise beaucoup de thread dans notre application. Pour connecter le socket, on a un thread, si non l’application va bloquer. Pour recevoir le message on doit créer un autre thread pour détecter le message de temps en temps. Pour recevoir les images, on a un autre thread en utilisant le protocole TCP/IP.

7.4 Résultat et Conclusion

Pour le moment, on a bien développé une application Androïde pour contrôler la voiture. Elle va marcher comme on veut. Même si il y a encore des bugs, ce n'est pas important. Pour nous, le plus important est que l'on a fait une chose de zéro à un, c'est-à-dire que l'on crée toutes les choses soi-même. Quand on voie la voiture roulant, on est vraiment heureux.

7.5 A faire plus loin

Pour un projet PRIM de deux semestres, on a fait beaucoup de chose, mais on est toujours curieux d’apprendre les nouvelles choses et améliorer notre projet. Car le temps est limite pour nous, nous listons des choses que l’on peut faire à l’avenir.

7.5.1 Contrôler la voiture par décider les paramètres en cote utilisateur

On peut ajouter des paramètres pour personnaliser de façon à contrôler la voiture.

7.5.2 Choisir la mode (trois mode possibles)

Maintenant on a juste deux façon de contrôler la voiture, manuelle et automatique. Mais on peut ajouter la troisième mode, c'est-à-dire l'utilisateur peut contrôler la voiture, mais quand la voiture détecte des obstacles ou d'autre chose, il va décider comment marcher mieux.

7.5.3 Compresser les images

Quand on envoie les images depuis la voiture à un portable, les images sont vraiment grosses, on peut compresser les messages quand on les envoie, et quand on reçoive, on les décomprime.

7.5.4 Optimiser l'algorithme de dessin

L'algorithme de dessin n'est pas encore optimisé, il faut faire mieux.

7.5.5 Embellir l'application

Maintenant on a juste implémenté la fonctionnalité pour l'application, mais elle ne semble pas joli, il faut lui embellir.

7.5.6 Faire l'app plus robuste

Parfois notre application va bloquer, on croit que le problème est que le socket se bloque, donc il faut le faire plus robuste.

8. Conclusion

Au cours de cinq mois de travail sur ce projet nous avons expérimenté le vrai challenge de construire un système embarqué d'assez grande complexité. L'union de la conception d'un SE traditionnel avec le développement FPGA nous a fait voir le monde du design et de la conception avec un nouveau regard. Nous avons appris comme les mécanismes de fiabilité dans un SE le rendent plus complexe, comme les accélérations hardware peuvent fonctionner dans une application réelle et quelles stratégies adopter. Nous avons mieux appris à mettre en place un système d'exploitation cohérent face à l'application attendue et à la customiser. Nous avons vécu l'expérience de faire la fusion de plusieurs domaines de connaissance dans un système embarqué à nous, et savoir identifier le réflexe des problématiques de chaque champ de connaissance dans le champ de l'électronique.

Par la suite de notre projet, on compte le diffuser sur internet sur la forme d'un toutorial. Nous croyons que ce projet la n'aurait pas pu être exécuté sans le soutien des exemples et l'inspiration trouvé sur d'autres projets diffusés sur le web.

Référence

- [1] https://reference.digilentinc.com/_media/zybo:zybo_rm.pdf
- [2] ZYBO-Embedded_Linux_Hands-on_Tutorial.pdf
- [3] <http://www.dbrss.org/zybo/tutorial4.html>
- [4] <http://www.instructables.com/id/Setting-up-the-Zybot-Software/?ALLSTEPS>
- [5] <http://javigon.com/2014/09/02/running-ubuntu-on-zynq-7000/>
- [6] <http://blog.csdn.net/lzy272942518/article/details/46663233#comments>
- [7] OV7725 Datasheet Version 1.4.pdf
- [8] http://www.xilinx.com/support/documentation/ip_documentation/v_vid_in_axi4s/v3_0/pg04_3_v_vid_in_axi4s.pdf
- [9] http://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf
- [10] Cours d'Athens, interface_user_space.pdf, Guillaume Duc
- [11] http://docs.opencv.org/2.4/doc/tutorials/imgproc/table_of_content_imgproc/table_of_content_imgproc.html#table-of-content-imgproc
- [12] https://en.wikipedia.org/wiki/Canny_edge_detector
- [13] http://www.xilinx.com/support/documentation/application_notes/xapp890-zynq-sobel-viva-do-hls.pdf
- [14] http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug902-vivado-high-level-synthesis.pdf
- [15] http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug871-vivado-high-level-synthesis-tutorial.pdf
- [16] <http://www.micropik.com/PDF/HCSR04.pdf>
- [17] <http://silabs.org.ua/bc4/hc06.pdf>