



Projet tutoré VEGA

Voiture informatisée

Système embarqué ARMADEUS

DA SILVA COELHO Nicolas - PIAU Anthony
IUT BELFORT-MONTBÉLIARD
2013-2014

Remerciements

Nous aimerions remercier M. Gavignet pour nous avoir laissé la liberté du choix du projet, M. Gustin pour nous avoir accompagnés durant tout le développement du projet et M. Humbert ainsi que le département de mécanique pour la conception de supports ainsi que la conception des cartes électroniques.

Sommaire

I. Introduction

- A. Cahier des charges
- B. Présentation du projet
- C. Matériel

II. Le système Armadeus

- A. Présentation détaillée du système Armadeus
- B. Prise en main de l'Armadeus

III. Développement des fonctions

- A. Partie Moteur
- B. Partie Servomoteur
- C. Partie Xbee
- D. Partie Ultrason

IV. Hardware

- A. Carte Alimentation
- B. Carte Xbee
- C. Carte Optocoupleur

V. Software

- A. Déploiement d'un Firmware
- B. Logiciel APF51
- C. Précisions sur la communication Xbee
- D. Problèmes rencontrés

VI. Conclusion

VII. Annexes

1- Introduction

Aujourd'hui, le monde du multimédia devient présent dans de plus en plus de domaines, le besoin d'afficher des interfaces graphiques complètes, attractives et ergonomiques devient un vrai enjeu commercial. Par exemple, les nouveaux combinés (308 etc), les téléphones portables, les supports linux embarqués voient leur importance évoluer dans le monde du multimédia. Les possibilités supplémentaires par rapport à un PIC est le système d'exploitation qui offre pratiquement toute les possibilités d'un ordinateur, IHM, serveur WEB messagerie, Base de données, driver déjà existant pour de nombreux matériel.

A. Cahier des charges :

Contraintes spécifiques à l'environnement

Nous travaillons sur un système embarqué, nous avons donc une contrainte d'espace occupé par les composants et cartes électroniques, ainsi que d'autonomie du système.

Contraintes économiques

Aucun budget maximal n'a été fixé, cependant les commandes ont été réalisées avec un compromis entre budget et performances.

B. Présentation :

But du projet

Ce projet a pour but principal la mise en œuvre d'une voiture informatisée ou autonome pour une prise en main analogique-numérique ainsi que l'approfondissement des connaissances en programmation de bas niveau.

Le projet sera suivi par le cadre enseignant afin de partager les connaissances sur le système Armadeus.

C. Matériel :

- Armadeus (APF51)



- Châssis DCM précision



- Servomoteur futaba



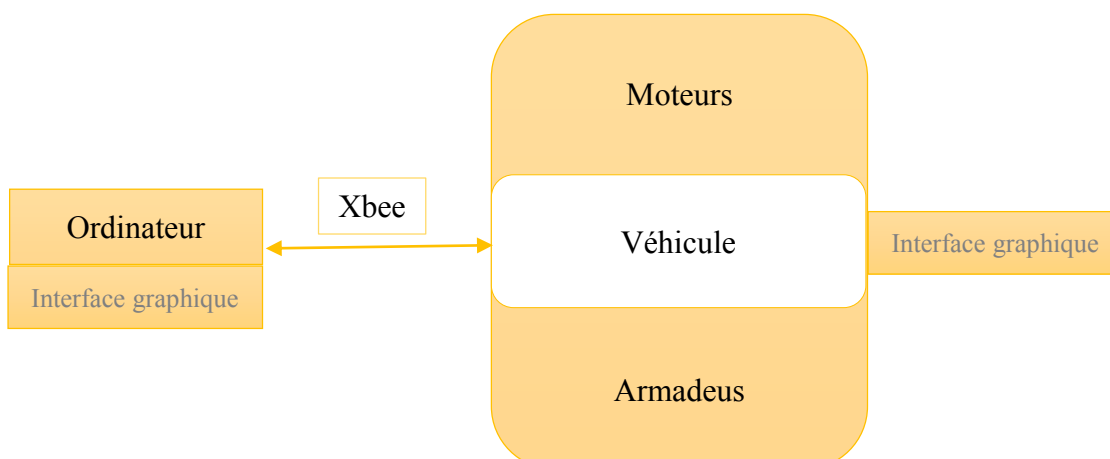
- Xbee



La conception de ce projet consistera à :

- Commander un moteur pour la propulsion en marche avant, arrière avec variation de vitesse
- Commander un servomoteur pour la direction
- Mettre en œuvre une communication Xbee pour la prise de contrôle à distance
- Visualiser les informations sur une interface Homme-machine
- Evoluer en automatisation complète du véhicule

Schéma fonctionnel 1er degré :



2- Le système Armadeus

A. Présentation détaillée du matériel Armadeus :

APF51 :

L'APF51 est une carte à microprocesseur de taille réduite bénéficiant d'un rapport coût/performance extrêmement compétitif.

Equipée d'un microprocesseur i.MX515 (ARM Cortex A8) à 800MHz, de 256 Mo RAM DDR Low Power, de 512 Mo de FLASH SLC NAND, d'un port Ethernet 10/100Mbps, de deux USB High Speed et d'un USB OTG, elle est facilement intégrable dans un système embarqué grâce notamment à ses régulateurs et ses convertisseurs de niveau (PHY RS232/USB/Ethernet).

Comme d'habitude chez Armadeus systems, cette carte est équipée d'un FPGA, ici un Spartan 6A XC6SLX9, ce qui augmente considérablement les fonctionnalités matérielles du module.

Remarque : aucun debugger externe (BDI, JTAG) n'est requis. Une simple liaison RS232 est suffisante pour des développements "faibles coûts". Un debugger Linux est disponible (GDB).



APF51-dev :

La carte électronique APF51_Dev est une plateforme de développement idéale pour l'expérimentation et la vérification d'applications Linux embarquées.

Elle permet d'accéder à toutes les fonctionnalités de la carte à microprocesseur APF51 tout en offrant un ensemble de solutions additionnelles telles que WiFi, une entrée micro, une sortie audio, une sortie TV numérique, un bus CAN, ainsi que le GPS et le GSM/GPRS/3G en option.

Des interfaces pour accueillir les modules CompactCom & AnyBus de la société HMS Industrial Networks SAS (bus de terrains maître ou esclave)

..... Permettent de créer facilement des prototypes d'équipements d'automatisme. Associée au runtime Straton (produit COPALP S.A.),

l'APF51_Dev permet même de concevoir des automates programmables extrêmement performants.

L'ensemble des drivers permettant d'utiliser les périphériques présents sur la carte sont disponibles dans le BSP Armadeus.

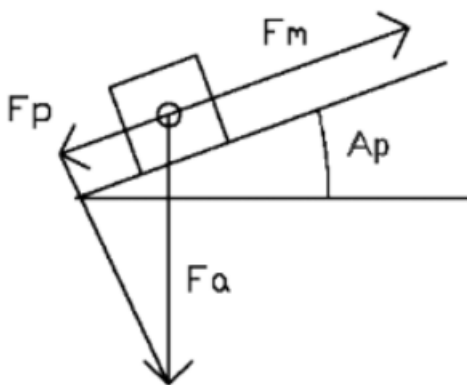


3- Développement des fonctions

A. Partie moteur :

Choix du moteur

Afin de pouvoir déplacer la voiture, nous avons recherché un moteur respectant un budget raisonnable et ayant assez de couple pour déplacer le véhicule sans d'importantes consommations.



Nous avons utilisés la formule :

Calcul du couple :

$$C_{\text{moteur}} = Pr \cdot D \cdot (Am + 9,81 \cdot \sin(Ap)) / (2 \cdot R) \text{ [Nm]}$$

- $Pr = 1 \text{ Kg}$
- $\sin(Ap) = 10/30$
- $D = 0.04 \text{ m}$
- $Am = 0.5 \text{ m/s/s}$

$$C_{\text{moteur}} = 0.04 \cdot 1 \cdot (0.5 + 9,81 \cdot (10/30)) / 4 = 0.0377 \text{ Nm} = 37.7 \text{ mNm}$$

Nous avons choisi le motoréducteur MFA série 919 avec les caractéristiques suivantes :

Alimentation: 4,5 à 15 Vcc

Consommation à vide:

- 0,45 A à 6 Vcc

- 0,52 A à 12 Vcc

Consommation en charge:

- 2,10 A à 6 Vcc

- 2,85 A à 12 Vcc

Réduction: 6:1

Vitesse: 2633 t/min à 12 Vcc

Couple: 741 g.cm

Diamètre de l'axe: 6 mm (avec méplat)

Embase de fixation: 49 x 47 mm

Dimensions: Ø38 x 101 mm

Poids: 234 g

Test du moteur

Nous avons ensuite pu tester le moteur à l'aide d'un module L298N pour gérer la marche avant et marche arrière du véhicule. Nous avons ainsi validé le test du moteur.

Support moteur

Afin de fixer le moteur sur la roue dentée du véhicule, nous avons dû faire un support pour surélever le moteur dans le but de le placer correctement.

B. Partie servomoteur :

Le servomoteur fournit avec le châssis du véhicule possède les caractéristiques suivantes :

Dimensions : 40.5x20x37.5mm

Poids : 55g

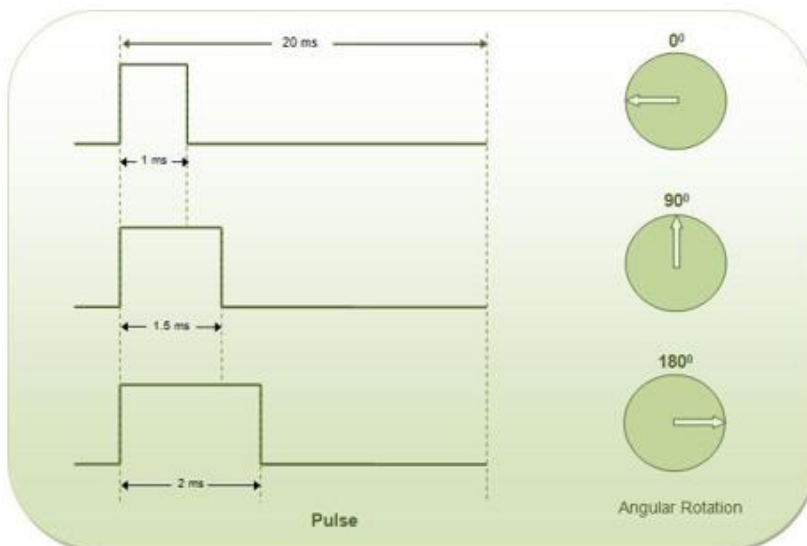
Couple : 8kg.cm

Vitesse : 0.10s/60°

Pignons aluminium

2 roulements

4.8 à 6V



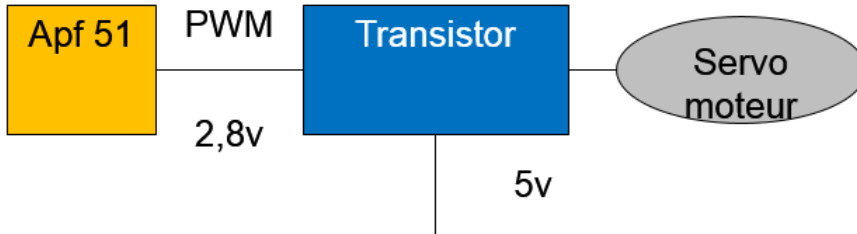
Test servomoteur

Un servomoteur est nécessaire au développement de ce projet afin de faire tourner le véhicule.

Pour tester le servomoteur, nous avons dû réaliser une adaptation car le signal PWM (modulation de largeur d'impulsion) produit par l'APF51 est de 0V à 2.8V alors que le signal PWM que doit recevoir

le servomoteur doit être de 0V à 5V.

Cette adaptation est faite par un transistor 2N2222 qui « tire » le signal PWM de 2.8V à 5V :



Nous avons relié le servomoteur à la direction du véhicule afin de procéder à des tests. On a validé ainsi le fonctionnement du servomoteur

Problèmes rencontrés :

Nous avons dû commander un pignon fusible après que celui-ci ai cassé à l'intérieur du servomoteur ayant trop forcé pour tourner les roues du véhicule. Nous avons donc réétudiés les angles pour la direction avant d'installer le nouveau pignon fusible dans le servomoteur.

C. Partie Xbee :

Pour notre projet, nous nous sommes tournés vers la version grand public du module Xbee plutôt que la version professionnelle aux mêmes performances excepté la portée qui ne nous sera pas utile car largement suffisante avec la version grand public.

Le module Xbee nous servira de moyen de communication sans fil entre l'ordinateur et le véhicule afin de pouvoir le piloter à distance.

Nous avons choisi ce module car il fonctionne dans la bande 2.4Ghz, est bon marché et consomme peu de courant.

Les deux modules Xbee doivent être les mêmes afin de pouvoir communiquer entre eux.

Test Xbee

Lors de nos premiers tests, nous avions un module Xbee grand public et un module Pro, nous n'avions pas réussi à les faire fonctionner dû à cette compatibilité.

Afin de tester et de valider le fonctionnement de ce dernier, nous avons ouvert une communication entre l'ordinateur et la voiture et vérifié si on recevait des données. Nous avons ainsi pu recevoir le signal d'une touche appuyée sur le clavier.

D. Partie ultrason :

Choix du composant

Après avoir réalisé un mini projet de programmation en langage C sur un capteur ultrasons, nous avons décidé de choisir le même module que pour le mini projet du fait de sa facilité d'utilisation ainsi que notre expérience sur ce capteur suite au mini projet.

Nous avons donc utilisé le capteur SRF02 :

Alimentation - 5v

Courant - 4mA

Fréquence - 40KHz

Range - 15cm - 6m.

Modes de connexion –

1 Standard I2C Bus.

2 Serial Bus

Unités – distance reportée en uS, mm ou inches

Poids faible - 4.6gm

Petite taille - 24mm x 20mm x 17mm

Problèmes rencontrés :

Le projet ayant été retardé pour cause de problème sur le matériel Armadeus ainsi que des autres divers problèmes, nous n'avons pas pu mettre en œuvre la partie capteur à ultrasons. Elle pourrait être envisageable dans le cas d'une évolution du système afin de le rendre autonome.

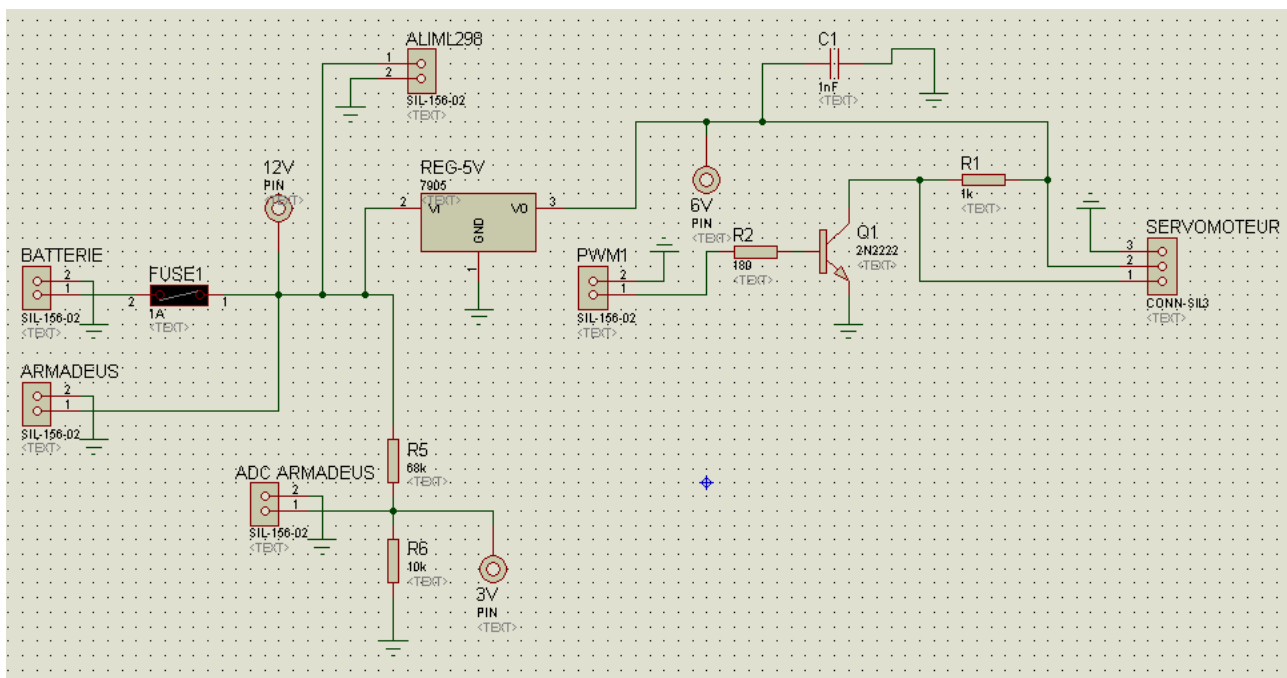
4- Hardware

A. Carte Alimentation :

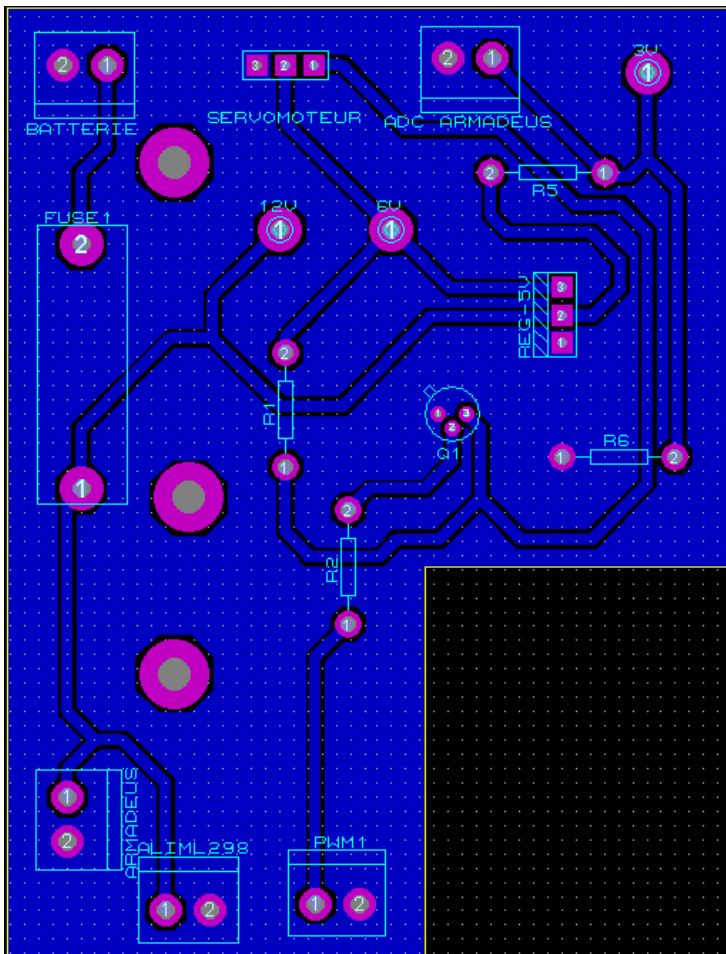
Après avoir choisi et testé le matériel un par un, nous devons l'alimenter. Les alimentations des différents composants étant différentes, nous avons donc dû réaliser une carte d'alimentation comprenant l'alimentation du servomoteur de 5V, l'alimentation du moteur de propulsion et l'alimentation pour la communication Xbee.

Nous avons alors choisit des composants pour répondre au cahier des charges :

- Un régulateur 12V-5V pour l'alimentation du servomoteur
- Un « pushpull » du signal PWM sortant de l'Armadeus à 5V pour qu'il soit exploitable par le servomoteur
- Un fusible
- Un pont diviseur de tension pour le convertisseur analogique numérique de l'Armadeus
- Ainsi que la répartition des alimentations par connecteurs

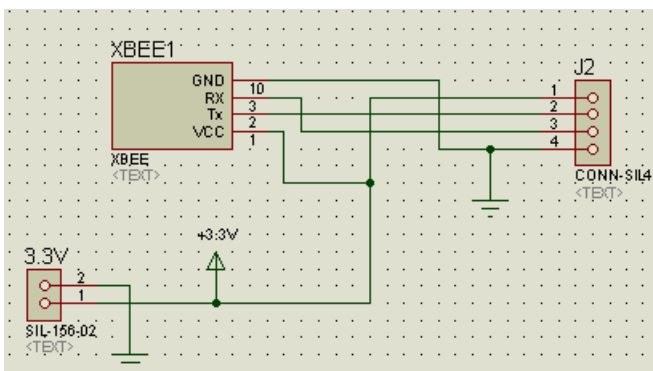


Pour le typon, nous avons dû adapter la carte au châssis du véhicule. Nous avons décidés de la placer en face du servomoteur sur la partie avant du châssis.

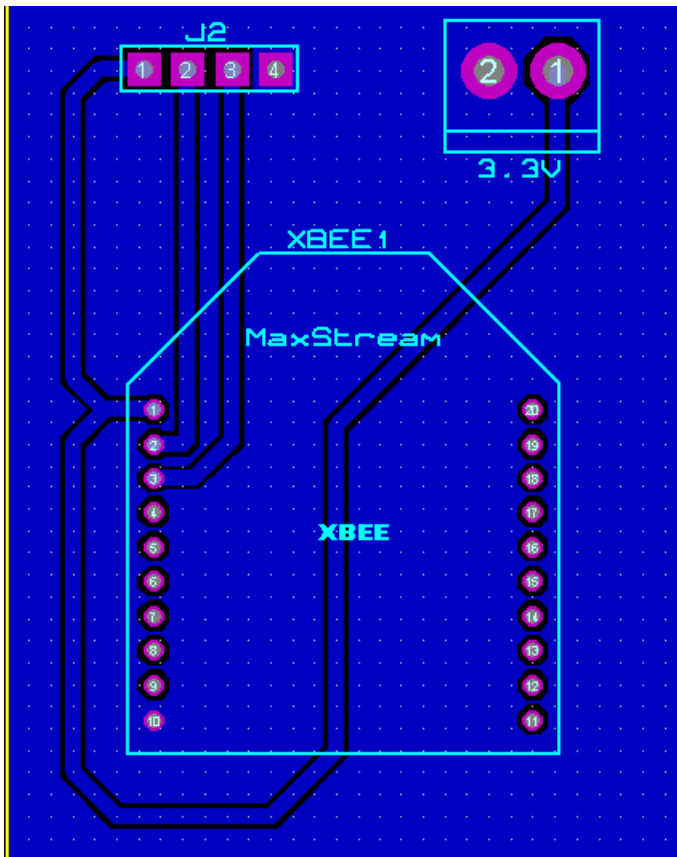


B. Carte Xbee :

Nous avons décidés de réaliser à part une carte Xbee afin de la placer au plus près de la carte Armadeus. Cette carte ne comporte que le module Xbee ainsi que deux connecteurs.

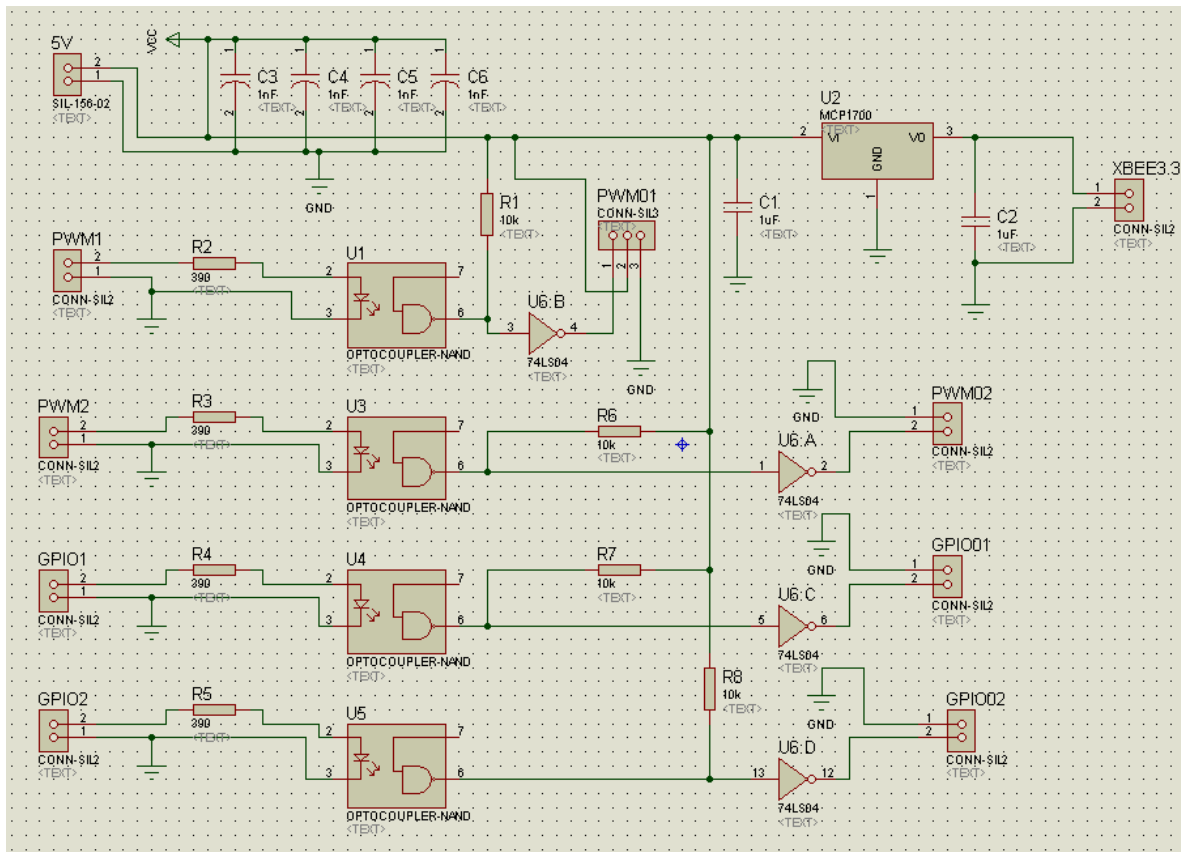


Typon :



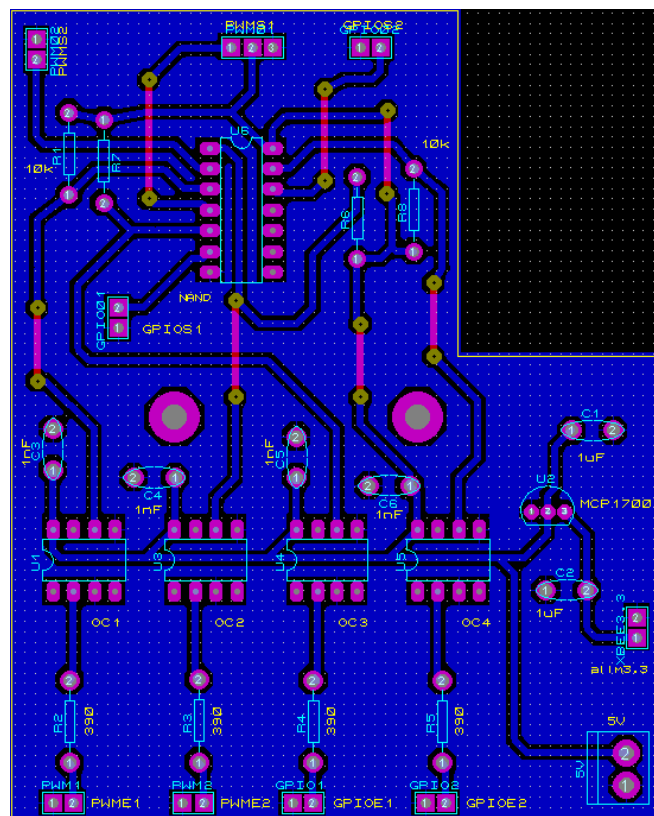
C. Carte optocoupleurs :

Après la réalisation de ces différentes cartes, lors des tests, nous avons eu un problème avec la carte Armadeus qui a subi une surchauffe ainsi que le module L298 du moteur. Nous avons donc choisit de faire une autre carte équipée d'optocoupleurs afin de couper les liaisons puissance-signaux. La puissance étant régulée et l'Armadeus ne pouvant plus consommer plus que ce qu'il ne devrait, le système fonctionne de nouveau.



Nous avons également intégré un régulateur pour le module Xbee afin d'annuler la consommation de courant depuis l'Armadeus.

Typon :



Nous retrouvons ainsi ces cartes sur le châssis de manière stratégique afin de pouvoir tout placer :



- En rouge : Carte d'alimentation
- En vert foncé : Carte optocoupleurs
- En vert clair : Carte Armadeus avec LDC
- En orange : Carte Xbee

5- Software

A. Prise en main de l'Armadeus :

Installation des outils de développement :

Prérequis :

Avant toute choses, il est nécessaire de posséder un ordinateur doté d'un système d'exploitation linux, dans notre cas une distribution opensuse. Pour la compilation du firmware et des outils de développement d'armadeus, il est nécessaire d'installer les paquets suivant en utilisant le gestionnaire de paquet « zypper »:

```
sudo zypper install gcc gcc-c++ autoconf automake libtool bison flex make
sudo zypper install subversion texinfo git-core ncurses-devel glibc-devel-static
sudo zypper install libacl-devel
sudo zypper install libuuid-devel
sudo zypper install libnetpbm-devel
sudo zypper install tcpd
sudo zypper install tcpd-devel
sudo zypper install lzo-devel
sudo zypper install tftp yast2-tftp-server
sudo zypper install python-serial python-usb
```

Installation des sources Armadeus :

On doit télécharger l'Armadeus software source code depuis <http://www.armadeus.com> → support
→ source code and patches

On télécharge le fichier « armadeus-5.2 .tar.bz2 » qui est un fichier compressé.

On décompresse le fichier en utilisant la commande « tar -xvf armadeus-5.2.tar.bz2 »

Un répertoire appelé armadeus -5.2 est créé et contient les sources du firmware, pour nous il se trouve dans /home/armadeus/Téléchargements/armadeus-5.2

On se place à la racine de ce répertoire puis il faut configurer le modèle de carte armadeus utilisée en tapant la commande « make apf51_defconfig ».


```
Terminal - armadeus@GE-EEP-08.iut-bm.univ-fcomte.fr:...nts/armadeus-5.2copy
Fichier Éditer Affichage Terminal Onglets Aide
armadeus@GE-EEP-08:~/Téléchargements/armadeus-5.2copy> ls
armadeus-5.2.tar.bz2  ChangeLog  Makefile      patches  software
buildroot            firmware   Makefile.in   scripts  target
armadeus@GE-EEP-08:~/Téléchargements/armadeus-5.2copy> rm armadeus-5.2.tar.bz2
armadeus@GE-EEP-08:~/Téléchargements/armadeus-5.2copy> ls
buildroot  firmware  Makefile.in  scripts  target
ChangeLog  Makefile   patches      software
armadeus@GE-EEP-08:~/Téléchargements/armadeus-5.2copy> make apf51_defconfig
mkdir -p downloads
wget --passive-ftp --tries=3 -P downloads http://buildroot.uclibc.org/downloads/
buildroot-2012.02.tar.bz2 \
|| wget --passive-ftp --tries=3 -P downloads ftp://ftp2.armadeus.com/arm
adeusw/download//buildroot-2012.02.tar.bz2
--2014-02-11 14:10:17-- http://buildroot.uclibc.org/downloads/buildroot-2012.02
.tar.bz2
Résolution de proxy-web.univ-fcomte.fr (proxy-web.univ-fcomte.fr)... 172.20.212.
238
Connexion vers proxy-web.univ-fcomte.fr (proxy-web.univ-fcomte.fr)|172.20.212.23
8|:3128...connecté.
requête Proxy transmise, en attente de la réponse...200 OK
Longueur: 2304351 (2,2M) [application/x-bzip2]
Sauvegarde en : «downloads/buildroot-2012.02.tar.bz2»

68% [=====] 1 572 208 155K/s eta 7s
```

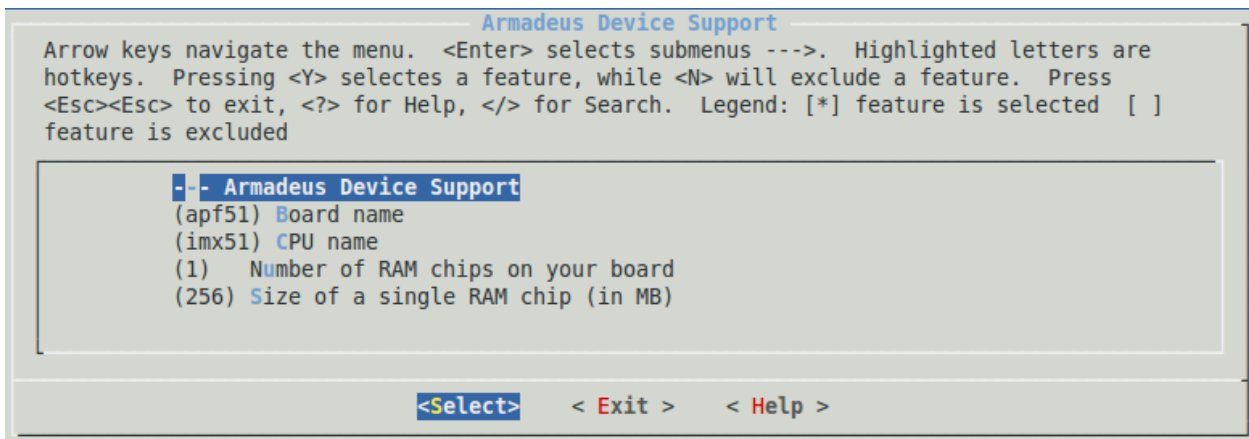
Un menu s'ouvre alors ou il faut vérifier si les configurations matérielles sont exactes en allant dans le menu

« System configuration -> Armadeus device support »

```
Terminal - armadeus@GE-EEP-08.iut-bm.univ-fcomte.fr:...nts/armadeus-5.2copy
Fichier Éditer Affichage Terminal Onglets Aide
/home/armadeus/Téléchargements/armadeus-5.2copy/buildroot/.config - Buildroot 2
0
Buildroot 2012.02 Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is
not selected

Target Architecture (arm) --->
  Target Architecture Variant (cortex-A8) --->
  Target ABI (EABI) --->
  Build options --->
  Toolchain --->
  System configuration --->
  Package Selection for the target --->
  Host utilities --->
  Filesystem images --->
  Bootloaders --->
  Kernel --->
  ---
  Load an Alternate Configuration File
  Save an Alternate Configuration File

<Select> < Exit > < Help >
```



Après vérification, nous pouvons quitter ce menu et valider les configurations. Sélectionnez les modules à utiliser, une connexion internet et nécessaire car, les modules sont téléchargés à la volée du au nombre important de modules disponible. Quittez et sauvegardez la configuration puis tapez la commande « make » qui lancera le téléchargement et la compilation de tous les modules, cette tâche est très longue (une à deux heures).

Modification pour Qt :

Comme précisé sur le wiki d'Armadeus, un changement d'url est a apporté sur le fichier « qt.mk » se trouvant dans l'arborescence buildroot/package/qt/qt.mk et apportez la modification suivante

```
# Replace line16:
QT_SITE      = http://get.qt.nokia.com/qt/source
# By:
QT_SITE      = ftp://ftp.qt-project.org/qt/source/
```

Modification pour pwm:

Encore une fois, comme précisé sur le wiki d'armadeus, les sorties PWM sont multiplexées un changement dans le fichier « apf51dev_baseboard.c » ce trouvant dans l'arborescence « buildroot/output/build/linux-2.6.38.8/arch/arm/mach-mx5/apf51dev-baseboard.c » est nécessaire il faut rajouter deux lignes en début du fichier :

```
#define APF51DEV_USE_LED_AS_PWM
#define APF51dev_USE_SWITCH_AS_PWM
```

Après ces changements il faut recompiler le noyau linux en utilisant les commandes :

« Make linux-clean » pour nettoyer les fichiers compilés.

« Make linux » pour recompile uniquement le noyau.

Dans le cadre de notre projet nous utiliserons la fonction SSH pour déployer notre projet : il faut donc rajouter le module OPEN SSH que nous trouverons dans : le menu

Package selection for the target -> networking applications -> open ssh

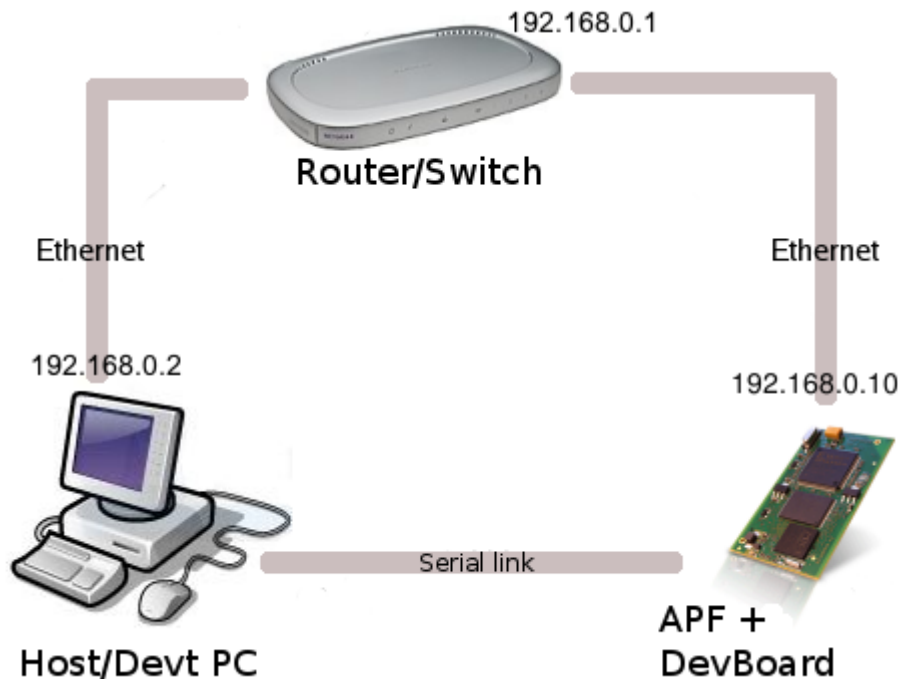
Ainsi que le package QT que nous trouverons dans :

Package selection for the target -> graphic libraries and applications -> Qt

Dans Qt, veuillez à bien sélectionner Gui Module.

Connexion de l'Armadeus :

Tout d'abord, reliez l'apf51 à votre ordinateur comme précisé sur le schéma ci-dessous :



La liaison sériel Link est une connexion USB sur le port « debug » de l'apf51
Ouvrir un terminal et taper la commande suivante :

```
armadeus@GE-EEP-08:~> minicom -D /dev/ttyACM0 -b 115200
```

Vous êtes à présent connecté au terminal de l'apf51

```
Terminal - armadeus@GE-EEP-08.iut-bm.univ-fcomte.fr:~
Fichier Éditer Affichage Terminal Onglets Aide

Bienvenue avec minicom 2.6.1

OPTIONS: I18n
Port /dev/ttyACM0

Tapez CTRL-A Z pour voir l'aide concernant les touches spéciales

U-Boot 2012.04.01 (Nov 07 2013 - 12:14:54) apf51 patch 1.7

CPU: Freescale i.MX51 rev3.0 at 800 MHz
Reset cause: POR
Board: Armadeus APF51 revision 1
DRAM: 256 MiB
NAND: 512 MiB
In: serial
Out: serial
Err: serial
Net: FEC
Hit any key to stop autoboot: 2
```

B. Déploiement d'un firmware :

TFTPboot est un serveur de fichier, c'est grâce à ce logiciel que l'apf51 peut récupérer les fichiers Firmware uboot et rootfs.

Tapez la commande « sudo yast2 tftp-server

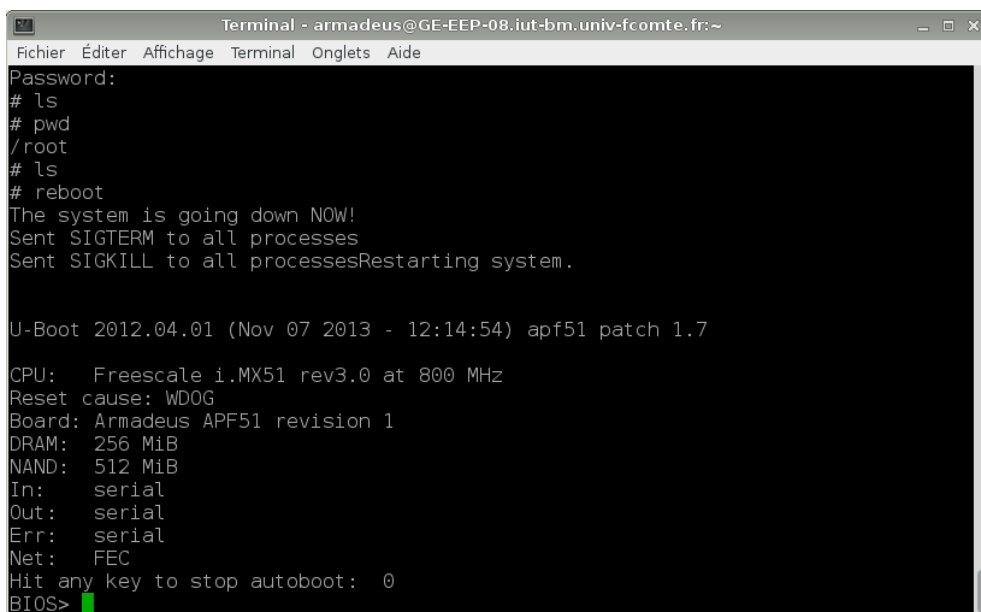
Choisissez « activé » pour ouvrir le port dans le par feu

Le point de montage par défaut et /srv/tftpboot.

Copier les fichiers compilés se trouvant dans buildroot/output/images/ vers /srv/tftpboot

Retournez sur le terminal de l'apf51.

Tapez la commande « reboot » pour redémarrer la carte, lorsqu'un décompte s'affiche, tapez sur n'importe quelle touche pour entrer dans le bios de l'apf51.

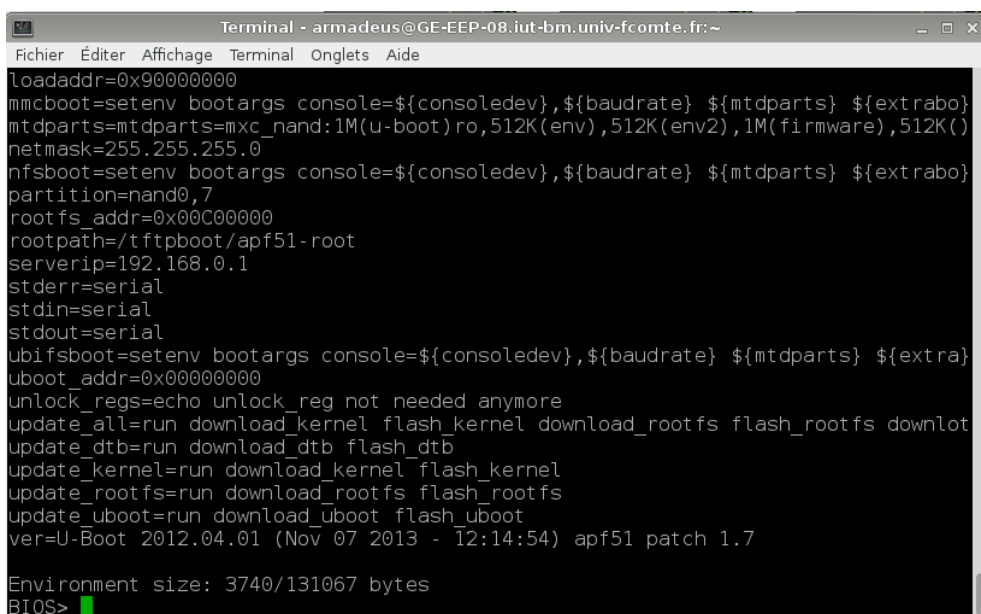
A terminal window titled 'Terminal - armadeus@GE-EEP-08.iut-bm.univ-fcomte.fr:~' showing the U-Boot boot process. The user enters 'reboot' and the system restarts. The U-Boot version is 2012.04.01 (Nov 07 2013 - 12:14:54) apf51 patch 1.7. The CPU is Freescale i.MX51 rev3.0 at 800 MHz. The board is Armadeus APF51 revision 1. The DRAM is 256 MiB and NAND is 512 MiB. The user is prompted to hit any key to stop autoboot, and the prompt BIOS> is shown.

```
Terminal - armadeus@GE-EEP-08.iut-bm.univ-fcomte.fr:~
Fichier Éditer Affichage Terminal Onglets Aide
Password:
# ls
# pwd
/root
# ls
# reboot
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processesRestarting system.

U-Boot 2012.04.01 (Nov 07 2013 - 12:14:54) apf51 patch 1.7

CPU: Freescale i.MX51 rev3.0 at 800 MHz
Reset cause: WDOG
Board: Armadeus APF51 revision 1
DRAM: 256 MiB
NAND: 512 MiB
In: serial
Out: serial
Err: serial
Net: FEC
Hit any key to stop autoboot: 0
BIOS>
```

Le bios contient de nombreux paramètre pouvant être affichés en utilisant la commande « printenv »

A terminal window titled 'Terminal - armadeus@GE-EEP-08.iut-bm.univ-fcomte.fr:~' showing the output of the 'printenv' command. It lists various environment variables for U-Boot, including loadaddr, mmcboot, mtdparts, netmask, nfsboot, partition, rootfs_addr, rootpath, serverip, stderr, stdin, stdout, ubifsboot, uboot_addr, unlock_regs, update_all, update_dtb, update_kernel, update_rootfs, update_uboot, and ver. The user is prompted to hit any key to stop autoboot, and the prompt BIOS> is shown.

```
Terminal - armadeus@GE-EEP-08.iut-bm.univ-fcomte.fr:~
Fichier Éditer Affichage Terminal Onglets Aide
loadaddr=0x90000000
mmcboot=setenv bootargs console=${consoledev},${baudrate} ${mtdparts} ${extrabo}
mtdparts=mtdparts=mxnand:1M(u-boot)ro,512K(env),512K(env2),1M(firmware),512K()
netmask=255.255.255.0
nfsboot=setenv bootargs console=${consoledev},${baudrate} ${mtdparts} ${extrabo}
partition=nand0,7
rootfs_addr=0x00C00000
rootpath=/tftpboot/apf51-root
serverip=192.168.0.1
stderr=serial
stdin=serial
stdout=serial
ubifsboot=setenv bootargs console=${consoledev},${baudrate} ${mtdparts} ${extra}
uboot_addr=0x00000000
unlock_regs=echo unlock_reg not needed anymore
update_all=run download_kernel flash_kernel download_rootfs flash_rootfs downlot
update_dtb=run download_dtb flash_dtb
update_kernel=run download_kernel flash_kernel
update_rootfs=run download_rootfs flash_rootfs
update_uboot=run download_uboot flash_uboot
ver=U-Boot 2012.04.01 (Nov 07 2013 - 12:14:54) apf51 patch 1.7

Environment size: 3740/131067 bytes
BIOS>
```

Tapez la commande « `setenv serverip 192.168.0.2` » puis « `saveenv` »

A présent, tapez la commande « `run update_all` »

Alors le firmware le rootfs et u-boot sont mis à jour via le réseau Ethernet.

Une fois la mise à jour terminée, tapez la commande « `reset` » pour redémarrer le système. Le noyau linux est chargé uniquement à la fin du décompte du bios.

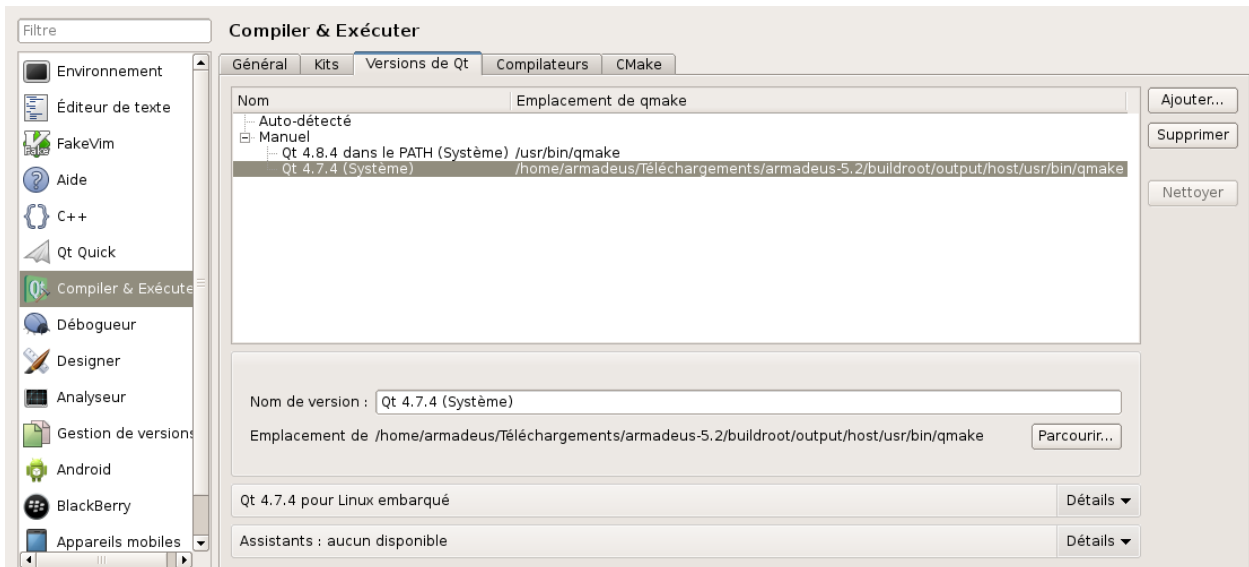
Nous utiliserons la library QT et l'IDE Qtcreator qui facilite la gestion de l'écran LCD le debugage et le déploiement d'application.

Lancer QtCreator puis aller dans Outils -> options

Tout d'abord il faut renseigner le compilateur Qmake :

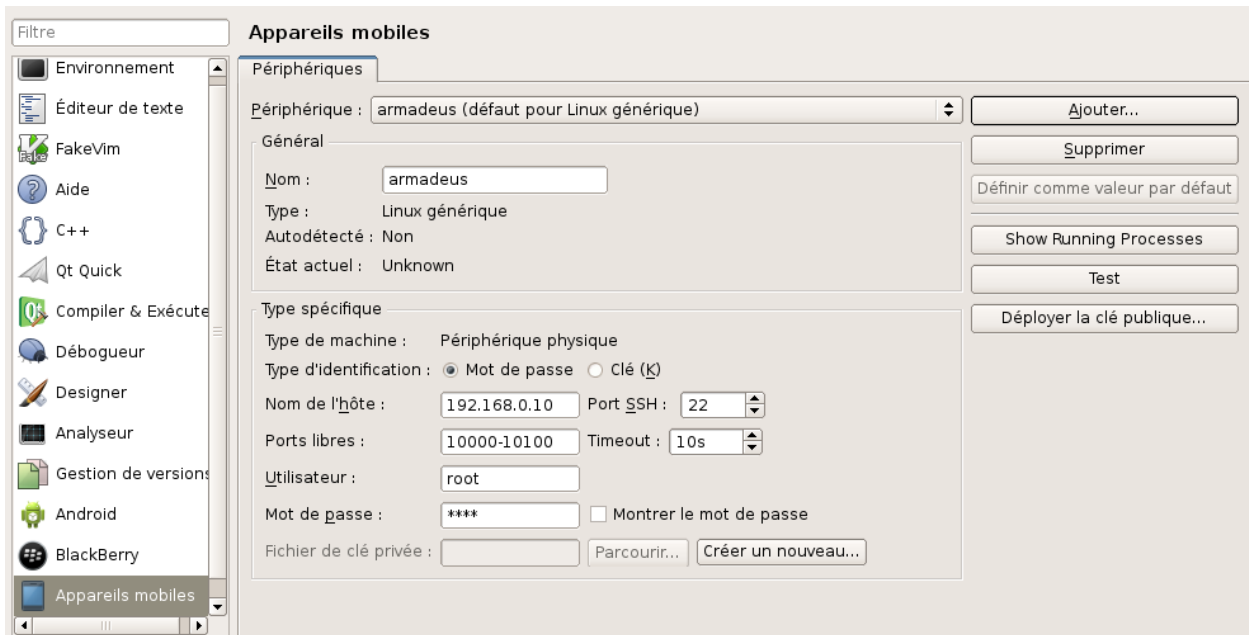
Aller dans « Compiler et Exécuter » puis dans l'onglet « versions de qt »

Cliquer sur ajouter et sélectionnez parcourir jusqu'à l'emplacement du fichier Qmake.



Ensuite aller dans « Appareils Mobiles » cliquez sur ajouter sélectionner périphérique linux générique

Ensuite rentrez l'adresse ip, le masque de sous réseau, le nom d'utilisateur et le mot de passe de l'apf51



Ensuite créez un projet Qtwidget :

Ouvrir le fichier .pro et rajouter les 2 lignes :

Target.path += « chemin du déploiement »

INSTALLS += target

A présent vous pouvez déployer l'application sur l'Armadeus automatiquement.

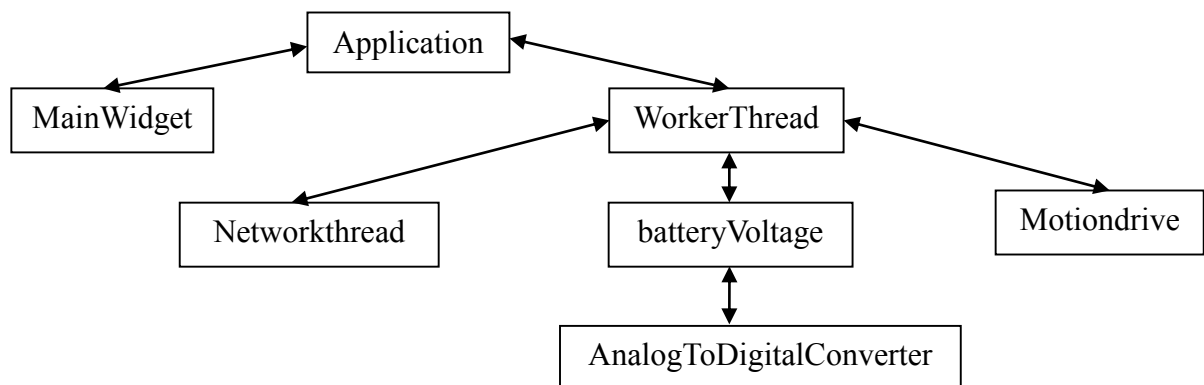
C. Logiciel APF51 :

D.

Architecture

Le logiciel de l'apf51 est programmé en c++ avec la library qt

Voici un schéma représentant l'architecture



Application : Permet de gérer les connexions de signaux et SLOT entre chaque classe.

MainWidget : décrit l’affichage d’une IHM sur l’écran tactile.

WorkerThread : Processus exécuté en parallèle afin d’éviter de figer l’IHM lors de traitement de données.

NetworkThread : processus exécuté en parallèle gérant les transferts réseaux via l’XBEE.

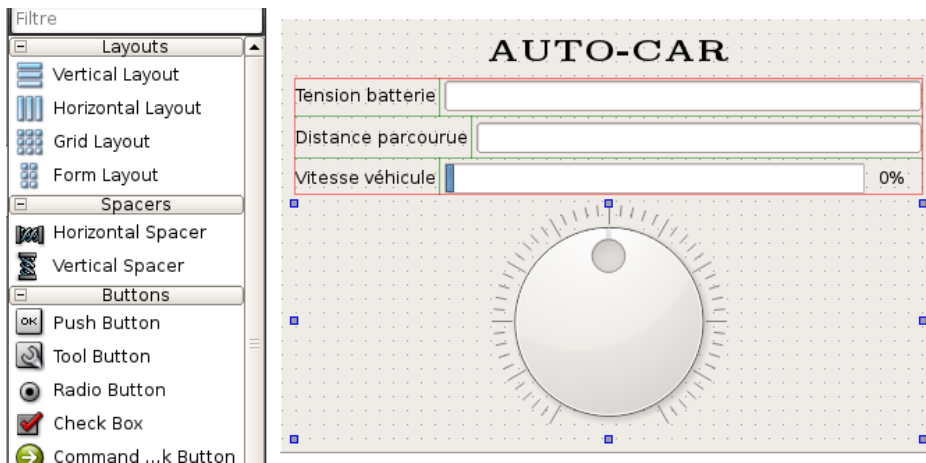
MotionDrive : classe permettant de gérer la marche avant/arrière et la direction de la voiture.

AnalogToDigitalConverter : classe permettant de gérer la conversion analogique numérique.

BatteryVoltage : c’est un namespace permettant de récupérer la tension de la batterie grâce à la classe AnalogToDigitalConverter.

Interface Graphique

Qt utilise un système de description d’interface graphique appelée QtDesigner :



Ce système consiste en une description écrite en XML

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3    <class>MainWindow</class>
4    <widget class="QWidget" name="MainWindow">
5      <property name="geometry">
6        <rect>
7          <x>0</x>
8          <y>0</y>
9          <width>470</width>
10         <height>310</height>
11        </rect>
12      </property>
13      <property name="windowTitle">
14        <string>MainWindow</string>
15      </property>
16      <layout class="QVBoxLayout" name="verticalLayout">
17        <item>
18          <widget class="QLabel" name="Title">
19            <property name="sizePolicy">
20              <sizepolicy hstretch="Preferred" vsizetype="Fixed">
21                <horstretch>0</horstretch>
22                <verstretch>0</verstretch>
23              </sizepolicy>
24            </property>
25            <property name="maximumSize">
26              <size>
27                <width>16777215</width>
28                <height>16777215</height>
29              </size>
30            </property>
31            <property name="font">
32              <font>
33                <family>PLRoman5</family>
34                <pointsize>16</pointsize>
35                <weight>75</weight>
36                <bold>true</bold>
37              </font>
38            </property>
39            <property name="acceptDrops">
40              <bool>false</bool>
41            </property>
42            <property name="text">
43              <string>AUTO-CAR</string>
44            </property>
45            <property name="alignment">
46              <set>Qt::AlignCenter</set>
47            </property>
48          </widget>
49        </item>
50        <item>
51          <layout class="QGridLayout" name="gridLayout">
52            <item row="0" column="0">
53              <widget class="QLabel" name="label">
54                <property name="text">
55                  <string>Tension batterie</string>
56                </property>
57              </widget>
58            </item>
59            <item row="0" column="1" colspan="2">
```

Ensuite, la suite Qt transforme ce fichier XML en un fichier .ui qui contient la mise en forme et les composants graphiques mais cette fois en langage c++ permettant de gérer les évènements générés par les outils d'interface et de renseigner les données à afficher.


```
26 QT_BEGIN_NAMESPACE
27
28 class Ui_MainWidget
29 {
30 public:
31     QVBoxLayout *verticalLayout;
32     QLabel *Title;
33     QGridLayout *gridLayout;
34     QLabel *label;
35     QLineEdit *lineEditBatteryVoltage;
36     QLabel *label_3;
37     QLineEdit *Distanceparcourue;
38     QLabel *label_2;
39     QProgressBar *progressBarSpeed;
40     QDial *dialGuidance;
41
42 void setupUi(QWidget *MainWidget)
43 {
44     if (MainWidget->objectName().isEmpty())
45         MainWidget->setObjectName(QString::fromUtf8("MainWidget"));
46     MainWidget->resize(470, 310);
47     verticalLayout = new QVBoxLayout(MainWidget);
48     verticalLayout->setSpacing(6);
49     verticalLayout->setContentsMargins(11, 11, 11, 11);
50     verticalLayout->setObjectName(QString::fromUtf8("verticalLayout"));
51     Title = new QLabel(MainWidget);
52     Title->setObjectName(QString::fromUtf8("Title"));
53     Title->setSizePolicy(QSizePolicy::Preferred, QSizePolicy::Fixed);
54     Title->setHorizontalStretch(0);
55     Title->setVerticalStretch(0);
56     Title->setHeightForWidth(Title->sizePolicy().hasHeightForWidth());
57     Title->setSizePolicy(sizePolicy);
58     Title->setMaximumSize(QSize(16777215, 16777215));
59     QFont font;
60     font.setFamily(QString::fromUtf8("PLRoman5"));
61     font.setPointSize(16);
62     font.setBold(true);
63     font.setWeight(75);
64     Title->setFont(font);
65     Title->setAcceptDrops(false);
66     Title->setAlignment(Qt::AlignCenter);
67
68     verticalLayout->addWidget(Title);
69
70     gridLayout = new QGridLayout();
71     gridLayout->setSpacing(6);
72     gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
73     label = new QLabel(MainWidget);
74     label->setObjectName(QString::fromUtf8("label"));
75
76     gridLayout->addWidget(label, 0, 0, 1, 1);
77
78     lineEditBatteryVoltage = new QLineEdit(MainWidget);
79     lineEditBatteryVoltage->setObjectName(QString::fromUtf8("lineEditBatteryVoltage"));
80     lineEditBatteryVoltage->setEnabled(true);
81     lineEditBatteryVoltage->setReadOnly(true);
82
83     gridLayout->addWidget(lineEditBatteryVoltage, 0, 1, 1, 2);
84 }
```

Les signaux et slots :

QT fonctionne sur le principe d'une boucle événementielle, lorsque qu'un évènement est activé, il est entré dans une PILE FIFO qui est parcourue en permanence.

```
→ Q_OBJECT

public:
→ explicit MainWindow(QWidget *parent = 0);
→ ~MainWindow();

public slots:
→ void setBatteryVoltage(int voltage);
→ void setSpeed(int speed);
→ void setGuidance(int guidance);
```

On déclare des slot à l'aide du mot clef « slots » mais la macro « Q_OBJECT » doit apparaitre dans la description de la classe sinon le slot n'est pas généré correctement dans la gestion des évènements.

```
→ emit batteryVoltage(res);
```

On déclare des signaux avec le mot clef « signals » de la même façon que les slots . L'utilisation est différente car les slots sont des fonctions « appelées » et un signal est un évènement qui appelle un slot, on utilise le mot clef « emit » pour émettre un signal.

```
→ QObject::connect(workerThread, SIGNAL(batteryVoltage(int)), mainwidget, SLOT(setBatteryVoltage(int)));
→ QObject::connect(workerThread, SIGNAL(speedCmd(int)), mainwidget, SLOT(setSpeed(int)));
→ QObject::connect(workerThread, SIGNAL(guidanceCmd(int)), mainwidget, SLOT(setGuidance(int)));
```

On utilise la fonction connect pour connecter les signaux et les slots entre eux, ensuite Qt gère automatiquement la boucle événementielle.

PWM

Pour utiliser les pwm on utilise une structure fournis dans le firmware QT dans le répertoire « as_device ».

```
as_pwm_device *device = as_pwm_open(engine);

as_pwm_set_frequency(device, 25000);
as_pwm_set_duty(device, realSpeed);
as_pwm_close(device);
```

Il suffit d'utiliser la fonction « as_pwm_open(numéro de la pwm) » pour instancier une structure correspondant à la pwm, ensuite des fonctions sont fournis pour gérer l'activation, la fréquence et le rapport cyclique.

GPIO

On utilise les GPIO d'une façon similaire aux pwm.

```
as_gpio_device *device = as_gpio_open(gpio);
as_gpio_set_pin_direction(device, "out");
as_gpio_set_pin_value(device, value);
as_gpio_close(device);
```

On utilise la fonction, « as_gpio_open(numéro de la pin) » ce qui instancie la structure as_gpio_device.

Ensuite nous avons des fonctions permettant de gérer si la GPIO est en lecture ou écriture, et des fonctions qui permettent de lire ou écrire son état logique.

ADC

Pour le convertisseur analogique numérique, il s'agit encore une fois des outils « as_device »

```
struct as_adc_device *adcDevice = as_adc_open_as1531(0, 2500);  
int adc = as_adc_get_value_in_millivolts_as1531(adcDevice, 3);  
as_adc_close(adcDevice);  
return adc / 0.126;
```

La fonction as_adc_get_value_in_milivolts_as1531() renvoi un entier correspondant à la tension lue en millivolts.

Lancement automatique

Pour le lancement automatique de l'application au démarrage il suffit de rajouter un script dans le répertoire « /etc/init.d/ » le nom du fichier doit commencer par SXX (XX = chiffre) le chiffre permet de donner une priorité à l'exécution du script, le fichier S02 est exécuté après S01.

Il suffit donc d'ajouter la ligne « /opt/autocar & » au fichier pour lancer le programme au démarrage de la carte.

E. Précisions sur la communication Xbee :

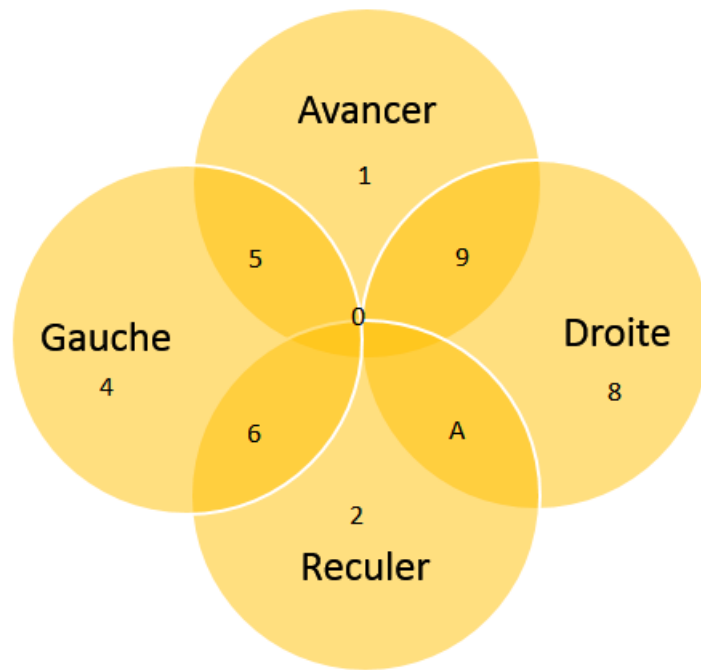
Le développement du software pour la communication Xbee est divisé en deux parties.

Une partie émission qui consiste à ouvrir un port de communication afin d'envoyer les instructions au véhicule telles qu'avancer, reculer, etc...

Une partie réception qui consiste à récupérer ces informations afin de mettre en marche les moteurs pour réaliser l'instruction envoyée depuis l'ordinateur.

Afin de pouvoir mettre ce système en place, nous devons réfléchir à un moyen d'envoyer plusieurs informations à la fois tel qu'avancer et tourner en même temps.

Nous avons optés pour une solution du type :



0 : Immobile
1 : Avancer
2 : Reculer
4 : Gauche
5 : Avancer et Gauche

6 : Reculer et Gauche
8 : Droite
9 : Avancer et Droite
A : Reculer et Droite

F. Problèmes rencontrés :

Tout d'abord, nous avons rencontrés plusieurs problèmes liés à l'installation de linux :

- Média d'installation de linux :
 - o Nous avons eu des problèmes par rapport à la capacité de la clé USB.
 - o Ensuite, il fallait la rendre « bootable » afin de pouvoir démarrer sur la clé USB pour l'installation. Nous avons dû utiliser plusieurs utilitaires afin de trouver le bon.
 - o Essai d'installation par netinstall : problème d'accès pour la configuration du proxy afin de procéder à l'installation.
- Disque dur :

Nous avons rencontrés des problèmes avec le disque dur reconnu dans le BIOS mais pas dans le programme d'installation de linux. Nous avons donc changé le disque dur afin qu'il soit reconnu.

6- Conclusion

Même si ce projet fonctionne et correspond au cahier des charges, il peut bénéficier d'évolutions afin de le perfectionner ou d'augmenter ses fonctions. On peut penser rajouter un capteur à ultrasons afin de détecter un obstacle devant le véhicule, plusieurs capteurs à ultrasons afin d'avoir une « vision 360° » autour du véhicule, développer l'interface homme-machine ou encore se focaliser sur une voiture entièrement autonome.