GRAND PROJET : DEVELOPPEMENT DE KART ELECTRIQUE

Etudiants:
Joseph KAPOU
Dimby Soa RAFALIMANANA

Responsable : M. Eric MELEIRO
Client : M. Christian PECOSTE

Travaux d'études et de recherche de Master GSAT/ISEE, Janvier à Mars 2014

REMERCIEMENTS

Nous exprimons notre reconnaissance à Christian PECOSTE, notre client. Son implication, sa disponibilité et sa simplicité sont autant d'éléments qui ont favorisé la réalisation de ce travail.

La qualité de ce Grand Projet fut enrichie par la contribution de Damien BLANCHARD. A qui nous tenons à exprimer toute notre gratitude pour les suggestions et conseils dans l'élaboration de la partie programmation de ce projet.

Nos sincères remerciements vont également à l'endroit de Eric MELEIRO qui nous a proposé le sujet de ce Grand Projet ainsi qu'à toute l'équipe pédagogique du Grand Projet E-Kart.

RESUME

Le projet E-Kart est un projet réalisé par plusieurs départements de l'Institut Universitaire et Technologique de Bordeaux 1. Il a pour objectif de fournir pour la rencontre pédagogique Nationale annuel de Vierzon, un Karting électrique fonctionnel.

Dans ce rapport, nous allons retracer notre participation au projet E-kart. Tout d'abord, nous allons présenter le contexte général du projet, les matériels utilisés notamment les Cartes Arduino, les capteurs ainsi que les logiciels. Par la suite nous allons présenter le système d'acquisition de données que nous avons élaboré. Ensuite, nous verrons tout ce qui concerne l'envoi des données au poste fixe au bord de la piste en temps réel. Ces données seront aussi envoyées à un module d'affichage afin que le pilote puisse aussi voir l'état du Kart. Et enfin, dans ce rapport nous verrons l'enregistrement des données sur une SD Card.

SUMMARY

E-Kart is a project realized by various departments of “Institut Universitaire et Technologique de Bordeaux 1”. It aims to provide for the annual meeting in Vierzon, a functional electrical Karting.

In this report, we trace our participation in E-kart project. First, we present the general context of the project, including materials used such as Arduino cards and sensors but also software used. Thereafter we will present the data acquisition system that we have developed. Then we will see data sending to the PC along the Karting track, in real time. These data will also be sent to a display unit to inform the driver about the Karting status. And finally, in this report we will talk about data recording on a SD Card.

TABLE DES MATIERES

INTRODUCTION

Chaque année, l'Université de Bordeaux inscrit dans son programme pour les étudiants au niveau Master en générale et en particuliers ceux deMaster Génie des Systèmes pour l'Aéronautique et les Transport (GSAT) ayant pour spécialité Ingénierie des Systèmes Électroniques Embarqués (ISEE), unepériode pendant laquelle les ceux-ci doivent, soit se retrouveren entreprisedans le cas d'un stage, soit faire des Grands Projets sur des thèmes proposés par l'établissement. C'est dans ce cadre qu'il nous a été proposé ce grand projet qui a pour thème: Développement sur kart électrique.

Ce Grand Projet d'une durée de 3 mois environ, dont deux mois et demi pour notre cas, s'est déroulé à l'Institut Universitaire et Technologique (IUT) sous la conduite deMonsieur Christian PECOSTE jouant le rôle du client.

Le thème de notre Grand Projet, fait partie d'un programme de rencontre nationale entre les IUT de France. D'où la participation de différents corps de métiers dans la réalisation dece projet dénommé « Projet E-Kart ».

Notre travail consiste à apporter une amélioration sur un autre Grand Projet du même thème déjà réalisé l'année dernière par les étudiants de Master 1, ISEE de l'Université de Bordeaux 1. Le travail qu'avaient effectué nos prédécesseurs sur ce projet, consistait à réaliser un Karting Électrique, que ce soit une modification d'un Karting Thermique, ou directement une conception. A cet effet, notre apport dans ce Grand Projet, concerne beaucoup plus la partie conception, notamment sur l'acquisition des mesures des capteurs et l'envoi des données par module XBEE.

Chapitre I: Présentation du Grand Projet

A. Généralité sur le projet

Comme nous l'avons évoqué auparavant, le Grand Projet qui nous a été confié s'inscrit dans le cadre d'un programme de Rencontre Nationale Pédagogique organisé chaque année par les IUT en France. Plusieurs équipes participent à la réalisation dudit projet selon leur domaine de compétence.

La répartition des tâches entre les différentes équipes s'étant fait de la manière suivante:

Les étudiants GACO sont chargés de l'aspect organisationnel. C'est à eux que revienne la gestiondes inscriptions de toutes les personnes désireuses de participer à ces quelques jours de rencontre pédagogique, en l'occurrence les étudiants et les enseignants. Ils sont chargés également de garantir aux participants les conditions d'hébergement, de transport entre autre.

Concernant la partie technique, plusieurs corps des métiers y ont été associés. Le domaine dela sécurité aux étudiants en Hygiène, Sécurité et Environnement, dont le respect des normes desécurité établit par le règlement de la rencontre E-Kart est une mission qui leur a été confiée, maisaussi l'organisation d'un stage de sensibilisation aux gestes des premiers secours. Les étudiants en Génie Mécanique sont chargés de gérer l'adaptation du châssis pour accueillir le nouveau moteur électrique ainsi que l'ajout des nombreuses batteries. Le calcul permettant d'utiliser les données envoyées par les différents capteurs est effectué par les étudiants en Mesures Physiques.A nous les étudiant en Master GSAT/ISEE, il nous a été demandé de concevoir un programme qui gèretraite les données des différents capteurs et de les enregistrer sur une carte SD. Ces données devraient être envoyées à travers une liaison sans fil sur un ordinateur au bord de la piste. Cependant les étudiants en GEII sont, quant à eux, charger de s'occuper de la partie qui concerne l'affichage de ces données envoyées sur un afficheur qui sera visible par le conducteur du Kart pendant la course.

Le diagramme suivant présente les deux entités qui interviennent sur le Projet E-Kart comme nous l'avons décrit ci-haut.

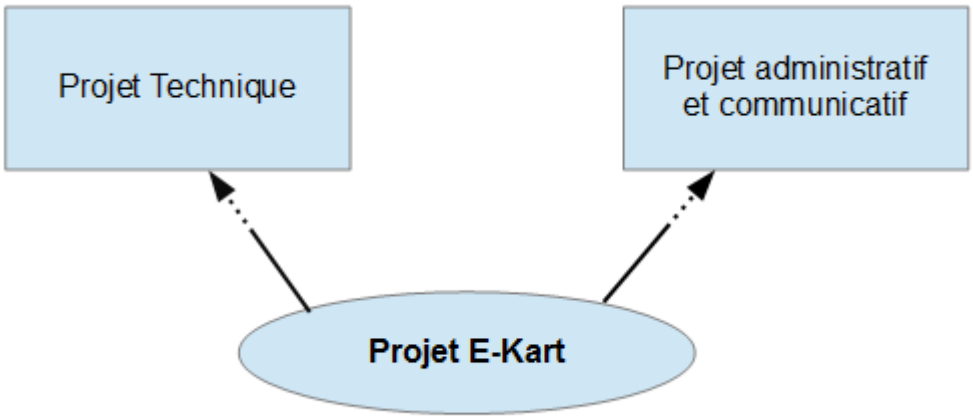


Figure 1 - Diagramme de description du projet E-Kart

Les trois équipes qui sont chargés de travailler sur la partie technique de ce projet se repartissent comme le montre le diagramme suivant :

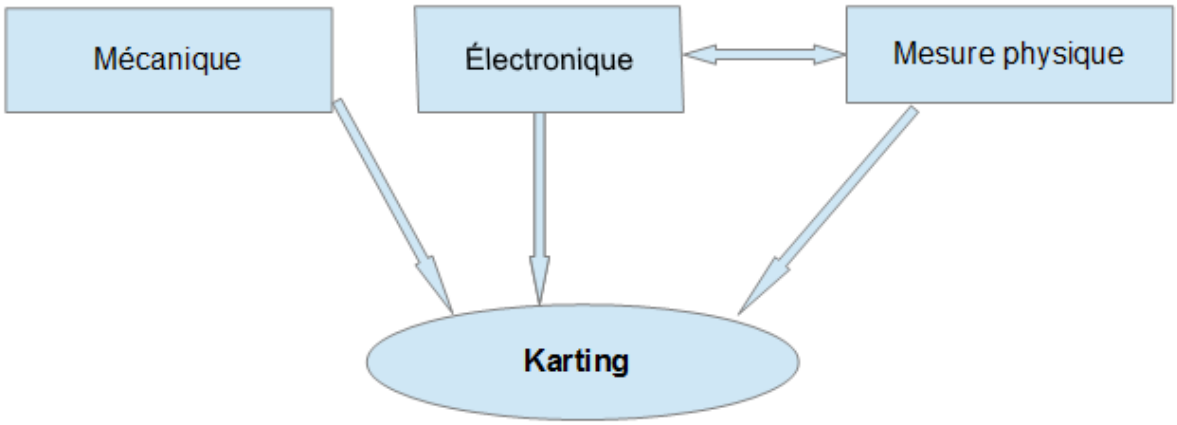


Figure 2 – Les parties prenantes du projet E-Kart

B. Projet tuteuré

Un thème quasi similaire à ce Grand Projet avait déjà, été traité par des étudiants en Master GSAT/ISEE de l'année dernière. Ce faisant, nous nous inscrivons dans la continuité de leur travail envue d'apporter quelques améliorations notamment au niveau de la conception. Notre tâche consiste à développer des programmes permettant de gérer l'acquisition les données des capteurs puis de les enregistrer sur une carte SD. Ces programmes permettront de déterminer la vitesse à laquelle se déplace la voiture, la température du moteur, le niveau de batterie, ainsi que les données du courant. Mais aussi cela permet de gérer le chronométrage qu'il effectue lors de son parcours et celui des tours précédents. Une fois que les données des différents capteurs seront connues, ces dernières devront être envoyées sur une carte SD ce qui permettra aux organisateurs de garder les

différentes informations (données) sur le Kart après la course. Ce faisant, à l'aide du logiciel informatique, nous aurons à envoyer les données des différentes capteurs un ordinateur sur le bord de piste.

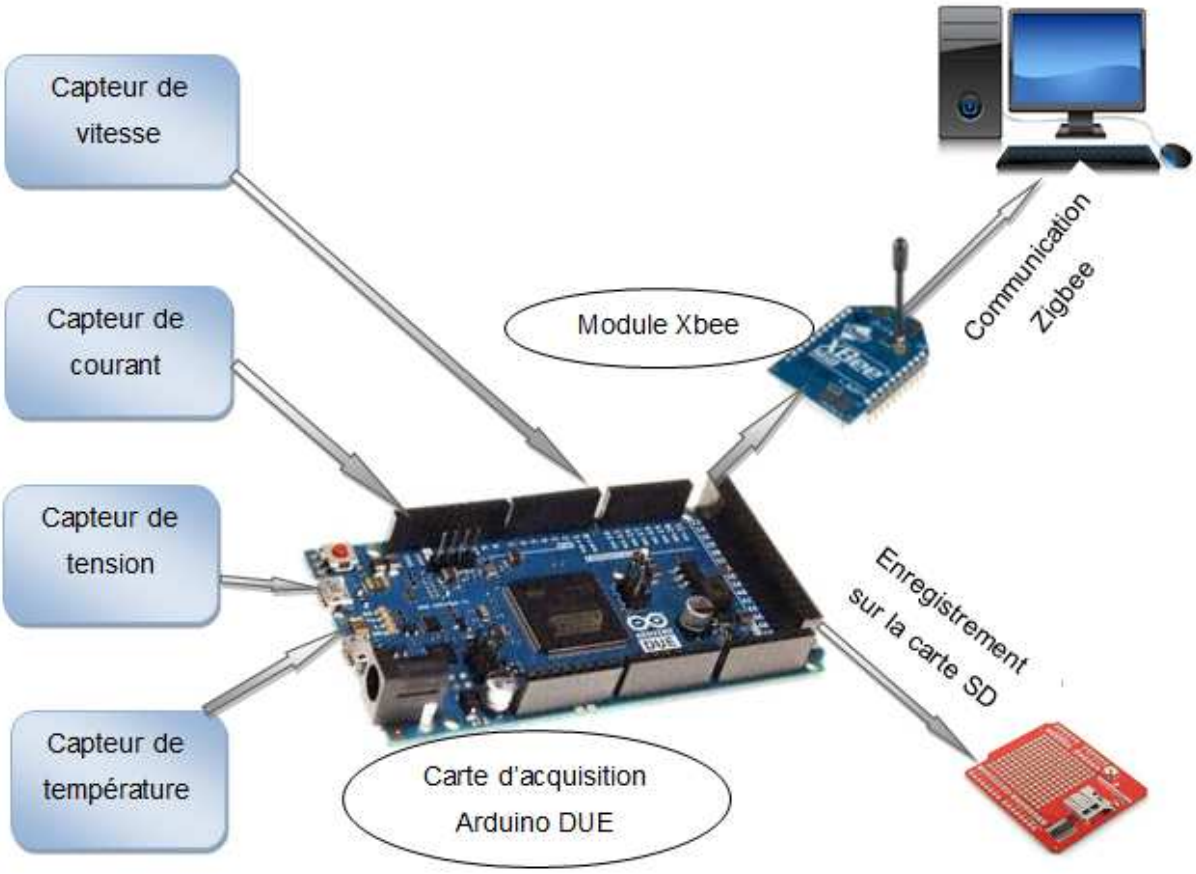


Figure 3 – Le système d'acquisition

C. Etudes de l'existant et notre apport

Contrairement à nos prédécesseurs de l'année dernière qui ont travaillé sur le même thème du projet à travers la gestion de la carte d'acquisition, la carte d'alimentation et la carte de la géolocalisation, notre tâche a été plus ou moins simplifié cette année car nous n'avons eu à gérer que l'acquisition des données. Les diagrammes suivants permettent de mettre en évidence les parties sur lesquelles les différentes équipent ont eu a géré.

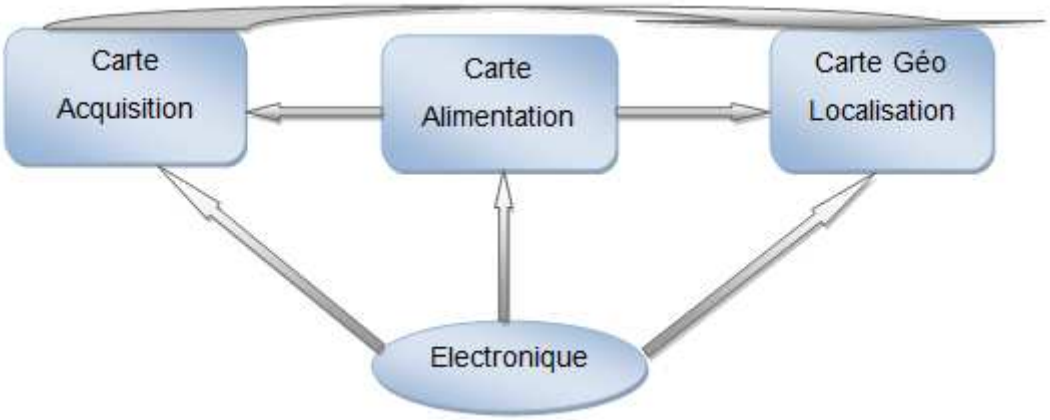


Figure 4 – Diagramme de Flux de Données – Partie électronique (Projet E-Kart 2013)

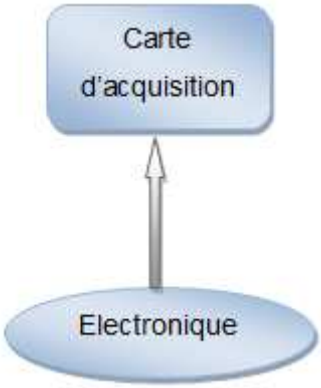


Figure 5 – Diagramme de Flux de Données (2) – Partie électronique (Projet E-Kart 2014)

Pour ce qui est de commun entre nous, à savoir la gestion de la carte d'acquisition des données des capteurs, nous avons dans un premier temps procédé à une étude sur ce qui avait été réalisé avant d'en apporter une amélioration.

Les quelques critiques que nous avons pu apporter sur leur travail sont les suivantes :

- Mauvaise gestion de l'espace mémoire de la carte SD. Les données étant enregistrées sans tenir compte du nombre exact d'octet à stocker. Le fichier qui permettait le stockage de ces données n'étant fermé qu'après chaque 5ms, ceci ne permettait pas de savoir si toutes les données à enregistrer y étaient.
- Leur diagramme de temps n'étant pas optimal, parce qu'ils utilisaient des délais pour la fréquence de chaque instruction. Cela est dû aussi au fait, qu'ils ont écrit tout leur programme qu'en scrutation.

Cependant, nous avons eu a apporté ce qui suit :

Nous avons fait une étude un peu approfondie sur l'enregistrement des données sur la carte SD. Ce qui nous a amené à :

Estimer un temps d'enregistrement pour un block.

Faire une gestion d'interruption toutes les 200ms

D. Cahier de charges

Concernant la partie qui nous a été demandé de réaliser sur ce Grand Projet, nous allons de manière succincte décrire les différents éléments qui y interviennent.

1. Présentation des capteurs

Pour notre grand projet, nous avons eu a travaillé avec deux types de capteurs, les capteurs numériques et les capteurs analogiques. Les capteurs utilisés dans le cadre de ce projet sont :

- Capteur de tension (analogique)
- Capteur de courant (analogique)
- Capteur de température (analogique)
- Capteur de vitesse (numérique)

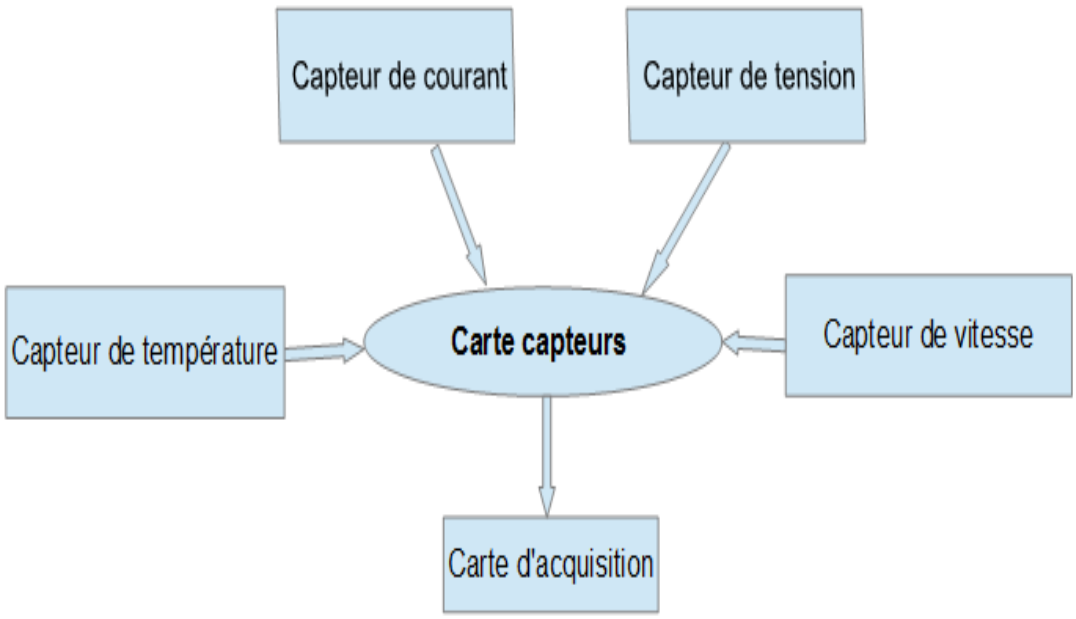


Figure 6- Les capteurs

a) Capteur de tension (capteur de supervision de batteries)

Figure 7 – Le capteur de tension

Ce capteur permet de lire la tension des batteries du Kart pour estimer à quel moment il faudrait les recharger par exemple. Le Karting est alimenté par 4 batteries, d'une moyenne de 12V, montées en série, ce qui, fait une tension globale de 48V, mais pouvant aller jusqu'à 52V directement après un chargement total. En considérant le poids du pilote ainsi que celui du Karting, nous constatons qu'une tension en dessous de 40V ne pourra faire circuler le véhicule, donc nous posons que :

40V=0% et 50V=100%.

Grace à la carte Arduino nous pourrons visualiser le changement de tensions entre 50V et 40V, mais il est impératif de l'adapter à l'entrée analogique de la carte vu qu'elle ne supporte que 5V. Un montage pont diviseur de tension simple est donc obligatoire, avec une tension de sortie environ 10 fois inferieure a la tension d'entrée.

Le diviseur de tension correspond au schéma ci-après.

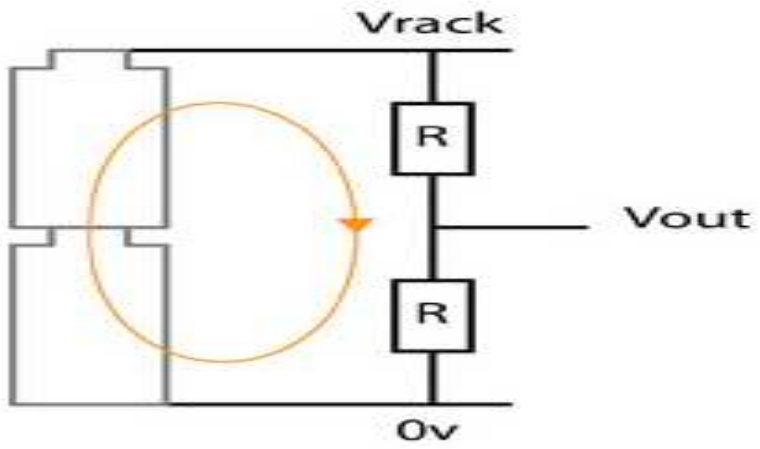


Figure 8 - Pont diviseur de tension

Ce circuit est simplement constitué de deux résistances en série (identique dans notre cas). ***Vrack*** est la tension aux bornes du ***rack*** batteries (6 piles de 1.2v en série soit 7.2v théorique). Le courant (orange) parcourt une seule maille composé des deux résistances, on a donc:

I = Vrack / (R + R) = Vrack / 2R (1)

Et Vout est donc donné par:

Vout = 0 + Ur = 0 + R*I (2)

Ce qui donne finalement en remplaçant I de (2) par (1):

Vout = Vrack / 2

L'intérêt de ce circuit étant qu'aucun courant ne sort par Vout et donc n'entrera par le port analogique de votre PRismino ce qui l'endommagerai sérieusement. De plus, quelque soit la charge mise en parallèle sur les batteries, Vout restera identique. C'est donc un bon moyen pour contrôler la tension de votre rack de batterie directement dans votre programme.

Ce montage consomme le courant I = Vrack / 2R, il est donc important de prendre de grandes résistances afin de ne pas décharger trop vite les batteries ou de griller les résistances avec un courant trop fort. Avec deux résistances de 10 kOhms on a un courant de 0.36 mA (milli-Ampères) ce qui est tout à fait négligeable par rapport aux moteurs (avec des piles de 2000mAh on peut donc alimenter le diviseur pendant plus de 5000h).

b) ***Capteur de courant***

- ***Exploitation du composant***

Selon le constructeur de ce capteur, pour la mesure du courant nous avons deux équations différentes à utiliser, selon le sens du courant, si le courant est positif, c'est à dire dans le sensconventionnel, cela signifie que le Karting avance, l'équation sera défini sous cette forme :

$$V_{out} = V_{ref} + \frac{I_{mes} \text{ x } 0.625}{I_{pn}}$$

Si le courant est négatif, autrement dit, le Karting recule, ce qui veut dire, au contraire, dans le sens conventionnel, l'équation devient alors :

$$V_{out} = V_{ref} - \frac{I_{mes} \text{ x } 0.625}{I_{pn}}$$

Avec Ipn=600 (fixé par le constructeur)

La valeur de Vrefest à fixer, pour être compatibles a la carte Arduino, qui prend au maximum 5V en entrée analogique, on choisit alors une tension Vref de 2,5V que nous allons avoir en mettant en œuvre un pont diviseur a résistances égales, la tension d'alimentation du pont sera la même que le Vcc du capteur, d'où les 2,5V obtenu. On prend deux résistances de 1kΩ pour le pont diviseur

Le schéma après l'ajout des résistances sera ainsi :

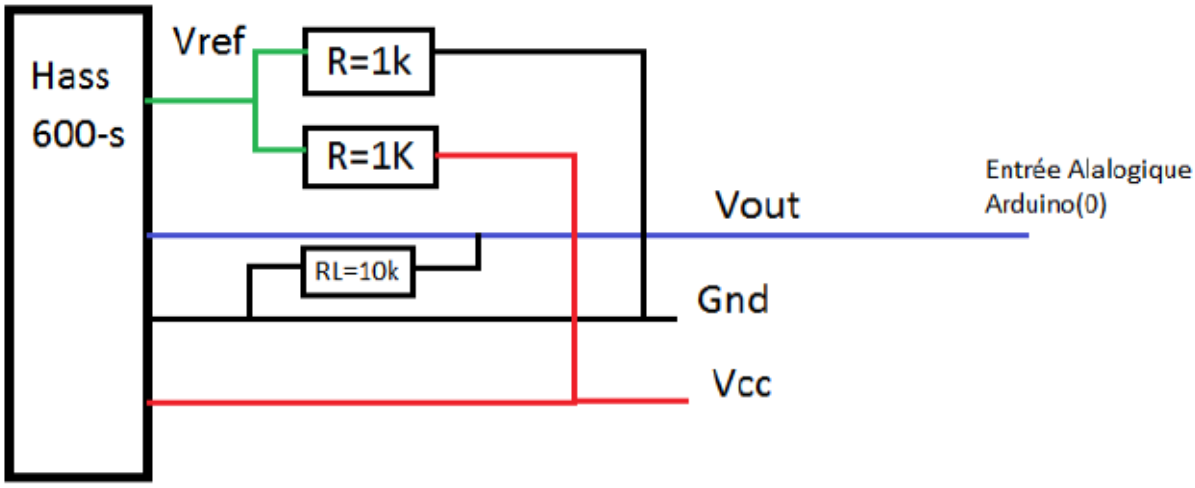


Figure 9 - Branchement du Capteur de Courant avec ajout des résistances

Apres avoir branche le capteur, on effectue une mesure de courant expérimental, on ne peut se fier directement à la courbe théorique, car les pertes ainsi que les parasites induits par un fort courant ne sont pris en compte.

Voilà quelques valeurs mesurées :

I mesuré (A)	$V_{out\ exp}$(V)	$V_{out\ théo}$(V)
-150	2,355	2,365
-130,4	2,378	2,386
-110,4	2,400	2,407
-90,2	2,423	2,428
-69,8	2,446	2,449
-49,8	2,468	2,470
-30	2,489	2,490
-10	2,512	2,511
-1,3	2,512	2,520
1,3	2,522	2,523
10	2,534	2,532
30	2,556	2,553
50,3	2,579	2,574
70,6	2,602	2,595
89,8	2,622	2,615
110,3	2,645	2,636
130	2,667	2,657
150,1	2,689	2,678

Figure 10 - Tableau de quelques valeurs de courant mesurées

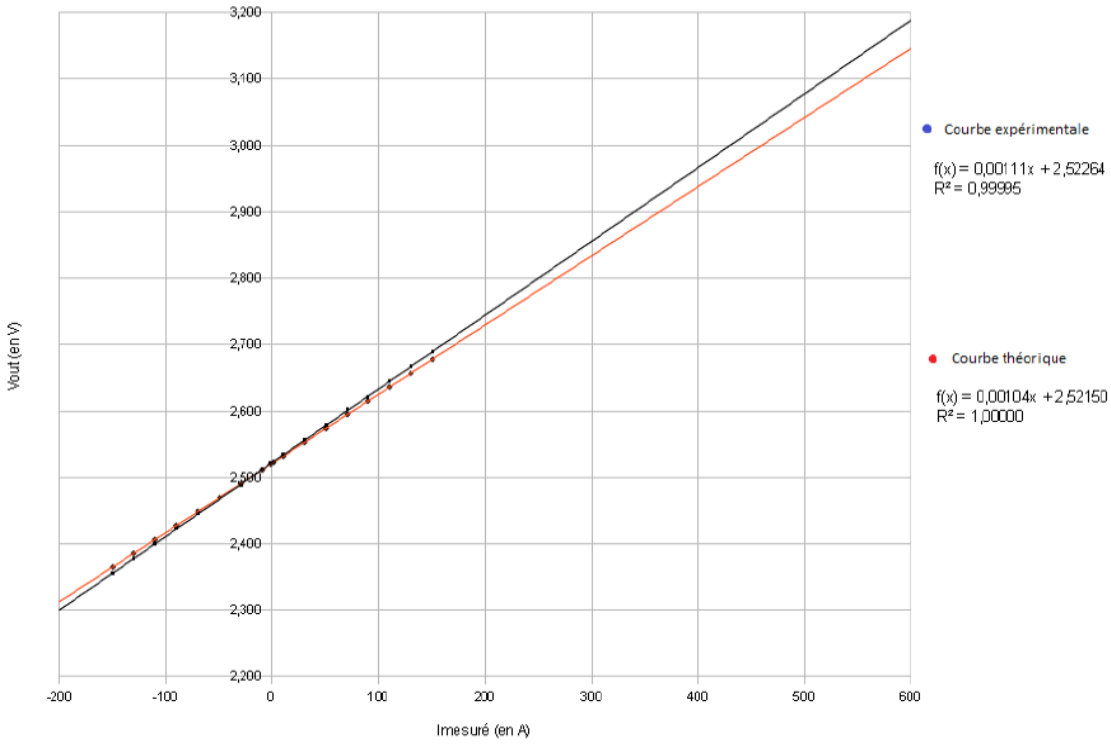


Figure 11 – Courbe d'étalonnage du capteur de tension

- ***Présentation du composant***

Figure 12 – Le capteur de courant

Le Capteur de courant HASS 600-s du constructeur LEM, est un capteur a effet HALL, lecourant mesure, induit un champ magnétique, qui entraine une tension induite sur la sortie. Utilise fréquemment pour des applications alimentées par batteries, convertisseurs statiques pour moteurDC.

- ***Caractéristiques du capteur***

• Principe de mesure a effet HALL

• Isolation galvanique entre l'enroulement primaire et l'enroulement secondaire

• Faible consommation d'énergie

• Alimentation Simple 5V

- ***Avantages du capteur***

• Facilite d'installation

• Faible encombrement

• Grande immunité aux perturbations

- ***Information des broches***

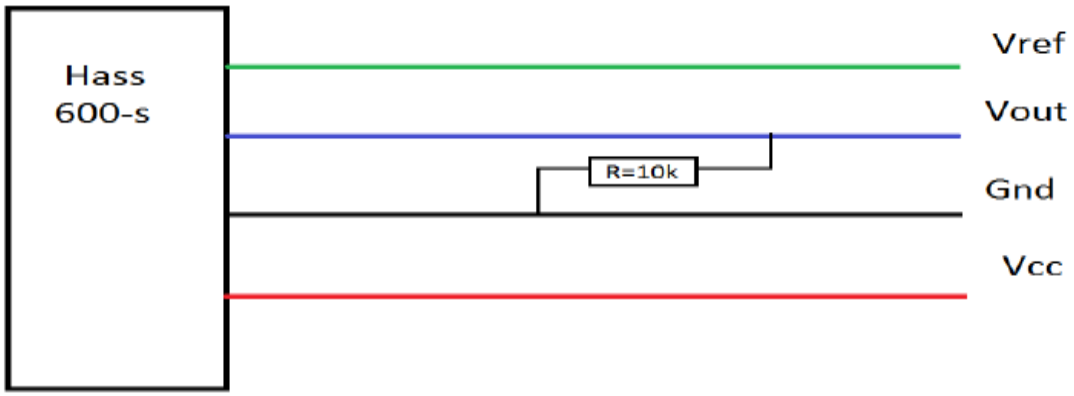


Figure 13 - Broches Capteur de Courant

c) *Capteur de température*

Le montage est sous forme d'un pont diviseur, la sortie est directement liée à l'entrée analogique de la carte Arduino.

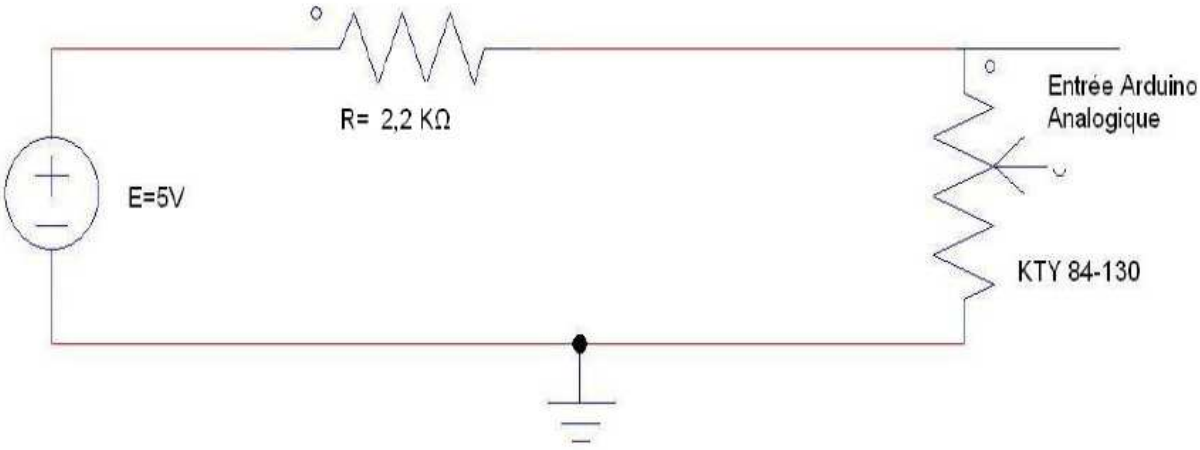


Figure 14 – Température

- ***Détermination de la valeur de R***

La détermination de la valeur de résistance et de la tension d'alimentation est faite en prenant en considération quelques contraintes. La tension maximum d'entrée de la carte Arduino est de 5V donc on impose une alimentation pour le montage qui ne dépasse pas cette tension.

Pour la résistance, on prend la valeur du capteur pour le minimum et maximum désire (0°- 100°), la relation pour calculer la tension au poin t entre les deux résistances nous permettra d'avoir la tension minimum et maximum. Nous avons teste plusieurs valeurs de R, et, après ces différents essais, nous avons remarqué que la résistance de 2,2 kΩ, délivrait un résultat exploitable :

$$V_{min} = 5\frac{498}{498 + 2200} = 0{,}92\,V$$

$$V_{max} = 5\frac{1000}{1000 + 2200} = 1{,}56\,V$$

La valeur de la résistance utilisée est celle qui nous permet d'avoir une marge assez importante, ici nous avons opté pour la 2200Ω donc

ΔV = Vmax-Vmin = 0,64

Comme cite ci-dessus, entre 0° et 100°, la caractéristique est qu asi-linéaire donc on se permet de choisir un pas de ΔV / 100, donc 0,0064 V/℃.

NB: on aurait bien pu prendre un montage compose d'un AOP non inverseur avec un gain de 3 ou l'entrée « + » est relie à la sortie du pont, qui nous permettra de diminuer la sensibilité. Par contre, dans ce cas, on ne peut choisir que la résistance de 2,2kΩ sinon la valeur maximum dépassera les 5V que la pin analogique Arduino peut supporter.

Vmax x 3 = 4,68V.

Avec une résistance de 1000Ω par exemple :

$$V_{min} = 5\frac{498}{498 + 1000} = 1{,}66\,V$$

$$V_{max} = 5\frac{1000}{1000 + 1000} = 2{,}5\,V$$

$$\Delta V = V_{max} - V_{min} = 0{,}84.$$

Et la Vmaxx 3 = 8,22V donc supérieur a 5V.

On pouvait aussi traiter toute la courbe du capteur en prenant l'équation de la CTP, mais vu que le Microcontrôleur en fin de projet doit gérer plusieurs capteurs ainsi la liaison Xbee, il n'est pas nécessaire d'exploiter des ressources et faire des calculs plus complexes qui peuvent ralentir le traitement des informations.

Donc on confirme le choix de la résistance 2200Ω.

- ***Présentation du composant***

Figure 15 – La thermistance

Le capteur de température KTY 84-130 a un coefficient de température positif (CTP), et est utilisé pour les mesures et les contrôles de systèmes. Le capteur est mis dans un boitier SOD68 (DO-34).

- ***Caractéristiques du capteur***

• stabilité à long terme
• gamme de température : -40°à +300°
• grande précision et fiabilité
• caractéristiques quasi-linéaires

- ***Information sur les broches***

a : Anode
k : Cathode

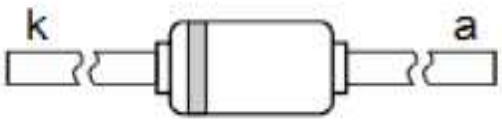
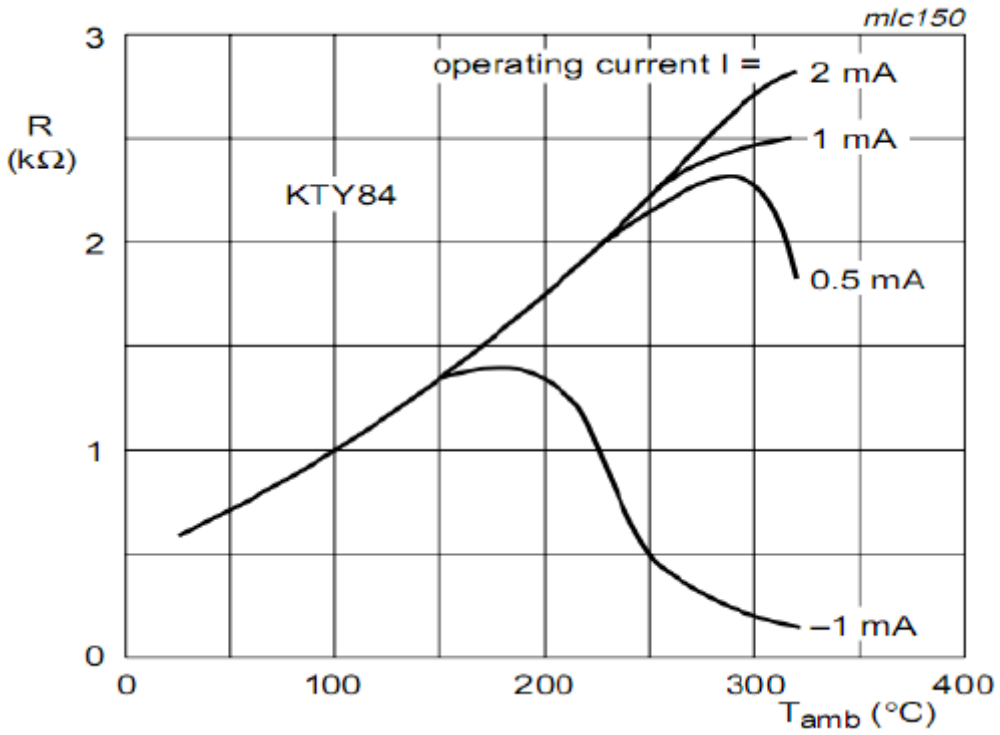


Figure 38 :Broche Capteur KTY 84-130

- ***Variation de la résistance en fonction de la température.***



Pour ce Grand Projet, les températures élevées ainsi que celles qui sont négatives importent peu, donc, nous avons choisi une gamme entre 0° et 100°, comme nous pouvons le voir sur le graphe, le morceau de la courbe entre ces deux valeurs est presque linéaire, le tableau ci-dessous donne la valeur exacte de la résistance en fonction de la température.

Température ambiante		**Résistance**		
℃	℉	Min	Typ.	Max
0	32	474	498	522
10	50	514	538	563
20	68	555	581	607
25	77	577	603	629
30	86	599	626	652
40	104	645	672	700
50	122	694	722	750
60	140	744	773	801
70	158	797	826	855
80	176	852	882	912

90	194	910	940	970
100	212	970	1000	1030

Figure 16 - Tableau de variation de la résistance en fonction de la température

d) Capteur de vitesse

C’est un capteur du type LJ12A3-4-Z/BY

Figure 17 - Capteur de vitesse

- **Présentation du composant**

Le capteur de LJ12A3-4-Z/BY est un détecteur de proximité inductif, lorsque le commutateur de proximité est proche d'un certain objet ≪ cible ≫, il envoie un signal de commande en forme d’impulsion. Ces composants sont souvent utilisés dans l'industrie de contrôle automatique de détection, de la commutation sans contact et des essais d'excès de vitesse.

- **Caractéristiques du capteur**

• diamètre de tête : 12mm
• distance de détection : 4mm
• tension d'alimentation : 6-36V DC
• courant de sortie : 300mA
• objet détecte : fer
• température de fonctionnement : -25°à +55°
• poids net : 46g
• longueur du câble : 110cm

- **Montage**

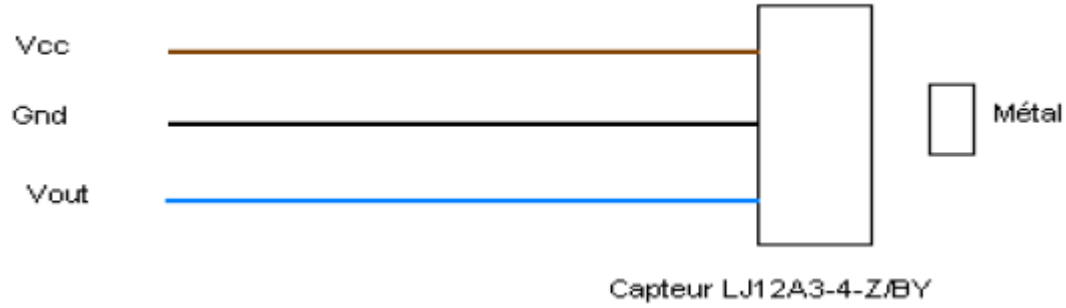


Figure 18 - Montage Capteur de Vitesse

2. Enregistrement des données

Pour garder la traçabilité des données reçues des capteurs lors de la course, il convient de les stocker dans un périphérique de masse. Et dans le cadre de notre Grand Projet, l'enregistrement se fera sur une carte SD. Cette dernière sera branchée sur la carte microcontrôleur Arduino Due à l'aide d'une liaison série. Les données à mémoriser seront stocké dans un fichier .TEXT. Dans ce fichier nous pourrons visualiser la forme de la trame envoyée.

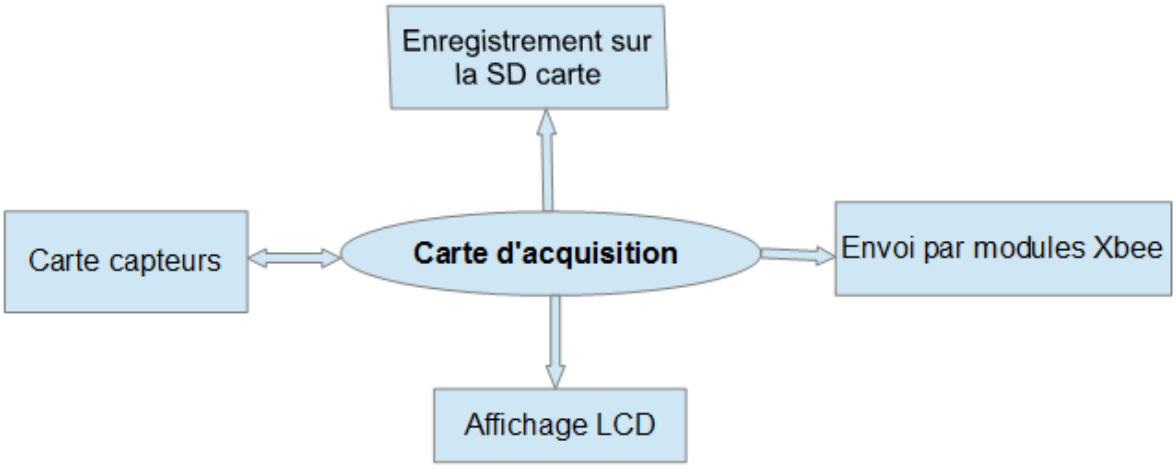


Figure 19 – Diagramme du fonctionnement global de l'acquisition

E. Présentation Matériels

Pour mener à bien notre Grand Projet, un certain nombre de matériels nous ont été mis à disposition par M. PECOSTE Christian, qui est notre client. Ces matériels sont :

– Deux Cartes Arduino Uno R3

– Une Carte Arduino Mega 2560

– Une Carte Arduino Due

– Deux Modules Zigbees

– Deux Modules Xbees

– 1 Module pour Carte SD 2 Go

– Trois Cordons d'Alimentation pour Carte Arduino ;

– Une Cartes de simulateur des capteurs ;

– Deux Platine pour Module Xbee

1. Les cartes Arduino

Dans le cadre de notre Grand Projet, nous avons utilisé les cartes Arduino. Arduino est une marque de cartes électroniques open-source qui possèdent un microcontrôleur programmable et basé sur une simple carte d'entrée-sortie et un environnement de développement qui met en œuvre le langage Processing. Arduino est certainement la technologie électronique la plus populaire auprès des passionnés mais aussi du monde de l'éducation. Les cartes Arduino sont d'excellents supports pour tout ce qui touche au prototypage et à l'expérimentation, faisant d'Arduino le matériel idéal pour un débutant.

Pourquoi Arduino ?

- Arduino est peu onéreux
- Arduino est multiplateforme
- Arduino propose un environnement de programmation simple et clair
- Le logiciel de programmation d'Arduino est gratuit et open-source
- Le matériel Arduino est open-source

a) la carte Arduino Uno

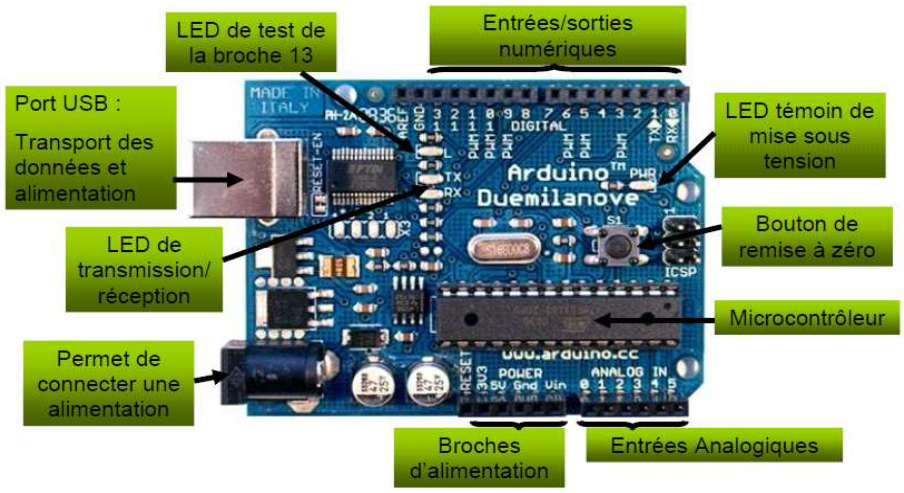


Figure 20 – Les différentes parties de la carte arduino UNO

Le modèle UNO de la société ARDUINO est une carte électronique dont le cœur est un microcontrôleur ATMEL de référence ATMega328 et une horloge cadencée à 16 MHz. C'est la plus récente et la plus économique carte à microcontrôleur d'Arduino. Des connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires. L'ATMega328 est un microcontrôleur 8 bits de la famille AVR dont la programmation peut être réalisée en langage C. Le contrôleur ATMega328 contient aussi un bootloader qui permet de modifier le programme sans passer par un programmateur.

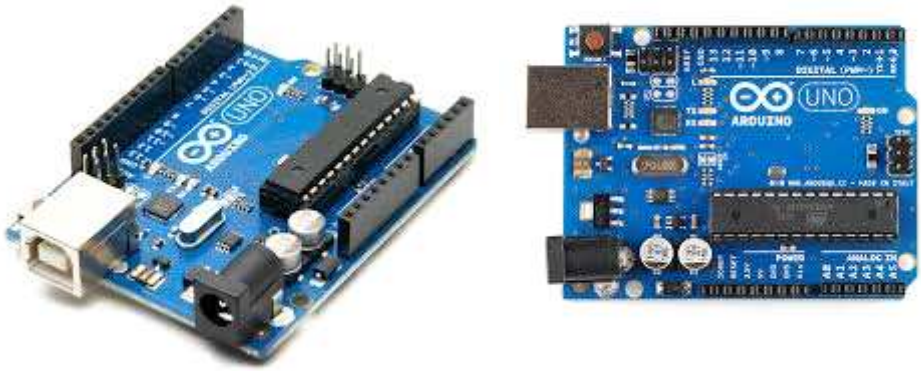


Figure 21 – La carte Arduino UNO

L'intérêt principal de cette carte ARDUINO (d'autres modèles existent) est sa facilité de mise en œuvre. ARDUINO fournit un environnement de développement s'appuyant sur des outils opensource. Le chargement du programme dans la mémoire du microcontrôleur se fait de façon très simple par port USB. En outre, des bibliothèques de fonctions sont également fournies pour l'exploitation d'entrées-sorties courantes : gestion des E/S TOR, gestion desconvertisseurs ADC, génération de signaux PWM, exploitation de bus TWI/I2C...

Cette carte est composée de 14 Pins Digitales Entrées/Sorties. Parmi ces 14 pins, nous avons 6 entrées analogique, et 6 autres qui peuvent être utilisé en sortie PWM, un port USB de connexion, une prise de courant, une embase ICSP, et un bouton poussoir servant pour le Reset. Cette carte n'utilise pas de puce de circuit FTDI USB-Série. Mais, elle dispose un Atmega16U2 (Atmega8U2 monte à la version R2).Pour l'alimentation de cette carte, il suffit de brancher un cordon USB ou un

adaptateur AC/DC, et, la source, est choisi automatiquement. Cette carte peut être alimentée par une source externe de 6 à 20V, dont, pour un bon fonctionnement il est conseillé par le constructeur d'alimenter de 7 à 12V.

Caractéristique de la carte Uno

Microcontrôleur	ATMega328
Alimentation	port USB ou 7 à 12V sur connecteur d'alimentation.Tension limite 6-20V
Broches E/S numériques	14 (dont 6 disposent d'une sortie PWM)
Broches d'entrées analogiques	6 (utilisables en broches E/S numériques)
Mémoire Programme Flash	32 KB (Atmega328) dont 0.5KB sont utilisés par le bootloader
Mémoire SRAM	2 KB (Atmega328)
Mémoire EEPROM	1 KB (Atmega328)
6 entrées analogiques	10 bits
Intensité maximale disponible pour la broche E/S (5V)	40 mA (200mA cumulé pour l'ensemble des broches E/S)
Intensité max. disponible pour la sortie 5V	Fonction de l'alimentation utilisée – 500mA si port USB utilisé seul
Intensité max. disponible pour la sortie 3.3V	50mA
Vitesse d'horloge	16 MHz
Bus série	I2C et SPI
Dimensions	74 x 53 x 15 mm
Gestion des interruptions	

b) La carte Arduino Due

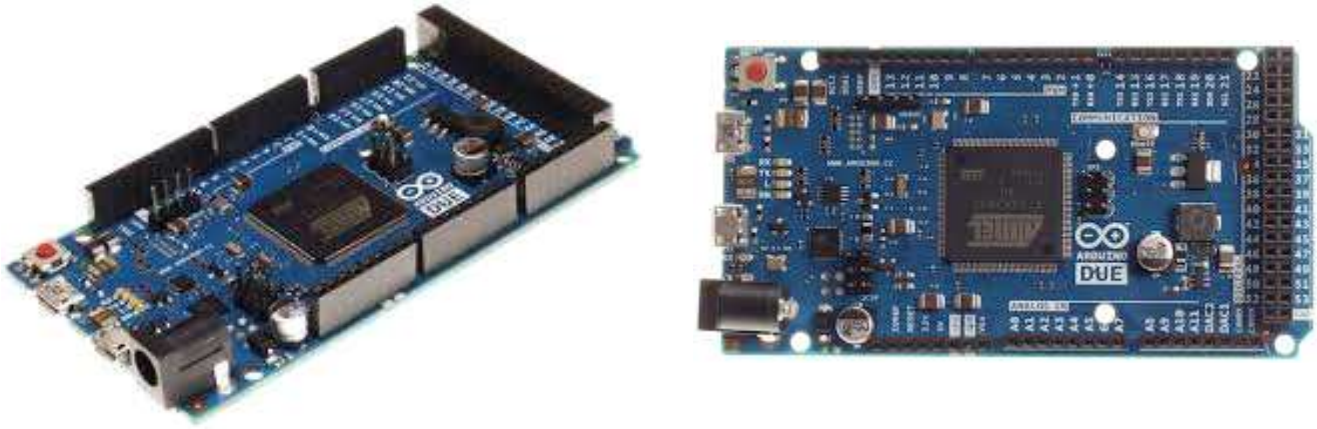


Figure 22 - La carte Arduino Due

Arduino DUE est la plus récente et la plus puissante des cartes microcontrôleur Arduino.Dotée d'un processeur ARM Cortex-M3 32 bits cadencé à 84MHz, ses capacités globales dépassent largement celles de ses illustres consœurs tout en préservant une compatibilité logicielle maximale.

A la différence des cartes basée sur AVR (5V) comme Arduino UNO, Leonardo ou Duemilanove, l'Arduino DUE fonctionne à une tension de 3,3V**.**Cependant, comme la carte DUE et toutes ses entrées / sorties fonctionnent à une tension de 3,3Vn il est fortement déconseillé de brancher de capteur ou un shield fonctionnant à une tension différente (5V par exemple) sous peine de gravement endommager la carte.

La compatibilité des shields est néanmoins assurée quel que soit la carte Arduino, 5V (UNO…) ou 3,3V (DUE) à la condition qu'ils respectent le format "1.0"; le pin "IORef" assurant alors la conversion de tension 3.3V / 5V.L'alimentation de la carte DUE demeure toujours aussi facile : micro-USB 5V ou connecteur Jack 7 à 12V - les convertisseurs de tension intégrés s'occupant du reste.

La carte Arduino Due accueille un microcontrôleur Atmel SAM3X8E ARM Cortex-M3. Elle possède 54 entrées/sorties numériques (dont 12 peuvent être utilisées pour une sortie PWM), 12 entrées analogiques, 4 ports série UART, une horloge à 84 MHz, une connexion compatible USB OTG, 2 DAC (digital to analog), 2 TWI, une prise jack, un header SPI, un header JTAG et enfin un bouton de reset et un bouton d'effacement.

Caractéristique de DUE

Nous mettons ici en exergue les caractéristiques techniques de la carte Arduino Due.

Microcontroleur	AT91SAM3X8E
Tension de travail	3.3V
Tension d'entrée (recommandé)	7-12V
Tension d'entrée (limites)	6-20V
E/S Numériques	54 (dont 12 avec sortie PWM)
Entrées Analogiques	12
Sorties Analogiques	2 (DAC)
Intensité de sortie totale sur toutes les E/S	130 mA
Courant max Pin 3.3V	800 mA
Courant max Pin 5V	800 mA
SRAM	96 KB (64 + 32 KB)
Vitesse d'Horloge	84 MHz

- **Le choix de la carte Arduino**

Dans la cadre de notre Grand Projet, nous avons eu à utiliser les deux cartes Arduino (la Uno et la Due). Vu la dimension de notre projet, il fallait faire un choix entre la carte Uno ou la Due. Et le choix final pour la mise en œuvre de notre projet s'est porté sur la carte Arduino Due pour des raisons suivantes :

- Comme sur l'Arduino Leonardo la fonctionnalité micro-USB host permettant l'émulation d'un périphérique comme un clavier ou une souris.
- Des performances de conversion analogique / numérique (ADC) considérablement améliorées : 12 pins ADC - 1000 Kbps (15 Kbps pour les cartes Mega, UNO et Leonardo)
- Un convertisseur de signal numérique vers analogique : une première sur une carte Arduino.
- La compatibilité avec le protocole ADK 2.0 : il est possible de connecter la carte DUE à un appareil Android en mode "Accessoire", comme sur la carte Mega ADK (Android Open AccessoryDevelopment Kit)
- La possibilité de jouer des fichiers son : une librairie permettant la lecture des .wav est disponible.
- Et encore 4 fois plus d'espace pour la programmation que sur les cartes Arduino Mega (16 fois plus que sur les cartes UNO).

Notre Critique : certes, la carte DUE est très performante et convient pour la gestion de nombreux capteurs et de calculs lourds (décodage audio, WiFi…) néanmoins, les projets plus simples ou débutants en électronique devraient s'orienter vers les Arduino Mega, UNO ou Leonardo, cartes éprouvées, stables et plus simples d'emploi.

2. Le modulemicroshield SD de lacarte SD

Ce module permet d'intégrer la carte SD qui nous permettra d'enregistrer les données des capteurs. Ce module est adaptable sur toutes les cartes Arduino, il suffit juste d'intégrer sa librairie dans ces dernières pour que cela fonctionne normalement. Pour son utilisation sur la carte Arduino, il convient de le brancher sur le port SPI, plus précisément sur les broches 8,11,12 et 13, qui correspondent respectivement aux SS, CLK, MOSI, MISO.

Pour notre Grand Projet, les données des capteurs que nous avons à enregistrer seront stocker dans un fichier TEXT car le fichier de format CSV parait lourd et du fait aussi qu''il faudrait l'ouvrir à chaque enregistrer. Or avec le fichier TEXT l'ouverture du se fait qu'une seule fois. Pour enregistrer ces données comme nous l'expliquons précédemment, cela se fera à l'aide d'un programme que nous allons écrire.

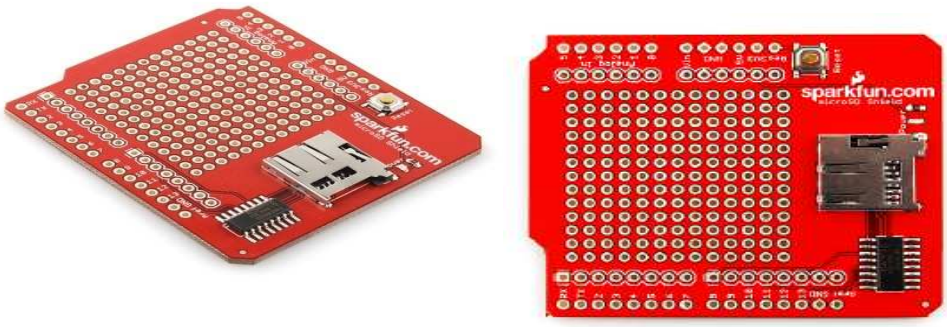


Figure 23 – Le microshield SD pour Arduino de chez Sparkfun electronics

3. Les Module Xbee

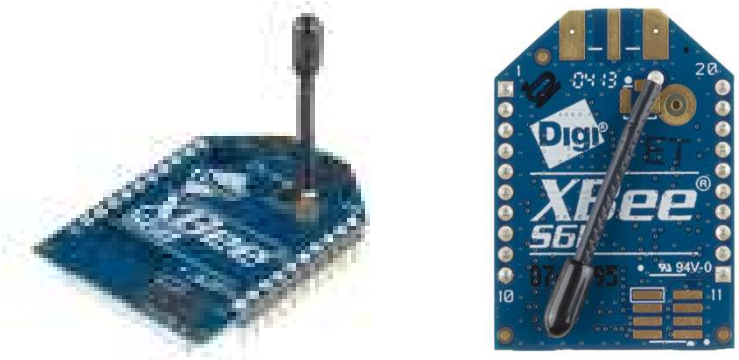


Figure 24 – Le module XBEE Pro

Les communications sans fils sont de plus en plus utilisées pour transmettre des informations entre différents systèmes électroniques. Pour les courtes distances et lorsque l'émetteur et le récepteur peuvent avoir un contact visuel, l'émission d'onde lumineuse infrarouge est souvent employée. Pour les communications sur de plus grandes distances impliquant des obstacles, c'est la communication par ondes radios (*RF*) qui est souvent privilégiée.

C’est pourquoi, dans le cadre de notre projet, nous avons utilisé le module XBEE qui permet de faire de la communication sans fil et celanous a permis d'envoyer des données des capteurs du Kart vers l’ordinateur au bord de la piste.

Ce module doit d’abord être configuré pour définir quel mode de communication l’on veut effectuer. Soit une communication porte-à-portesoit d’un module Xbee à plusieurs. Afin d’utiliser le module Xbee sur la carte Arduino, nous avons utilisé un autre module du nom de Zigbee, qui permet quant à lui, à relier le Xbee à la carte Arduino. Le Xbee peut aussi être utilisé sur une platine USB. Nous avons utilisé les deux méthodes de connexion du module Xbee dans le cadre de notre projet.

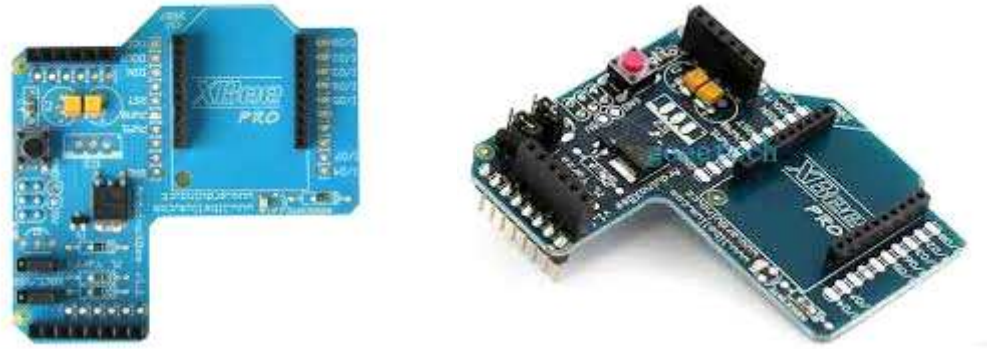


Figure 25- Le module Zigbee

Comme nous l'avons expliqué ci-haut, c'est ce module qui permet la connexion entre le module Xbee pro et la carte Arduino.

4. Le simulateur des capteurs

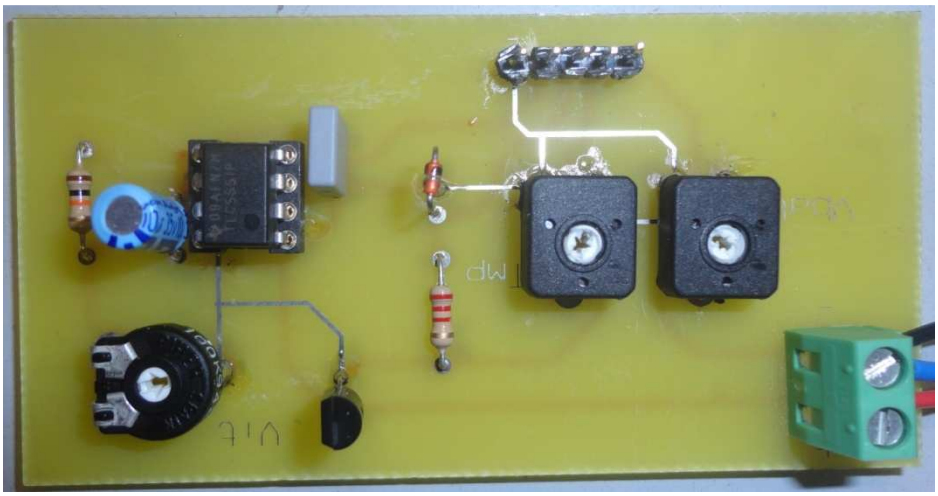


Figure 26 – La carte de simulation des capteurs

C’est une carte qui nous a permis de simuler les différents capteurs que nous avons utilisés dans le cadre de ce Grand Projet. Cette simulation des capteurs est un processus qui consiste à fournir des signaux réalistes identiques aux capteurs sur les entrées d'une unité sous test tout en évaluant la façon dont un élément de l'équipement réagit sur une large gamme de fonctionnements.

Le principal intérêt de simuler des capteurs est la capacité d'outrepasser les limites opérationnelles d'un environnement spécifique et de tester les conditions de rupture qui seraient, sinon, dangereuses ou dommageables, donnant à notre test une plus grande étendue avec un risque minimal. Ceci nous a permis aussi de pouvoir mettre en œuvre et tester des modifications apportées aux composants du système sans craindre de détruire un équipement coûteux. Les signaux simulés vont des simples formes d'ondes analogiques aux protocoles numériques personnalisés.

F. Présentation des logiciels utilisés

1. Logiciel Arduino

ARDUINO fournit un environnement de développement avec un éditeur de source, Les opérations de compilation et de chargement dans la mémoire du microcontrôleur étant ramenées à des clicks sur des boutons dans l'IHM (très simple). La communication entre le PC et la carte se fait via le port USB, moyennant installation d'un driver adapte (fourni par ARDUIINO).

Afin de pouvoir écrire nos programmes qui vont traiter les données des différents capteurs, nous avons utilisé le logiciel Arduino, qui est un logiciel propre à notre carte. Ce logiciel dispose d'un compilateur qui permet de vérifier les codes que nous aurons à écrire avant de les envoyer sur la carte. C'est un logiciel qui utilise un langage de programmation qui lui est propre, même si ce langage ressemble un peu plus a C++. Selon les besoins du programmeur ou selon le type de programme qu'il compte écrire, il est nécessaire de faire appel à des librairies spécifiques. Certaines librairies viennent par défaut avec le logiciel, cependant il n'est pas exclu d'en télécharger d'autres selon leur utilité.

Notons que ce logiciel est soumis à un certain nombre de règles que l'on doit respecter pour que le programme y marche correctement. L'usage des fonctions ≪**setup**≫ et ≪**loop**≫, s'avèrent indispensable. La fonction **setup** sert parfois à la déclaration des différentes variables cependant la fonction**loop** permet de boucler notre programme. Mais il est à noter que l'usage des fonctions setup et loop, n'exclut

de créer d'autres fonctions supplémentaires. Ces fonctions peuvent être nommées comme le veut le programmeur et elles peuvent être déclarées avant ou après la fonction loop. L'une des particularités de ce logiciel est qu'elle intègre en son sein, un terminal qui permet déjà au programmeur de visualiser le résultat de son programme ou en d'autres termes de communiquer avec les cartes sur lesquelles l'on a envoyé les programmes.

L'interface de ce logiciel à l'ouverture, se présente comme suit :

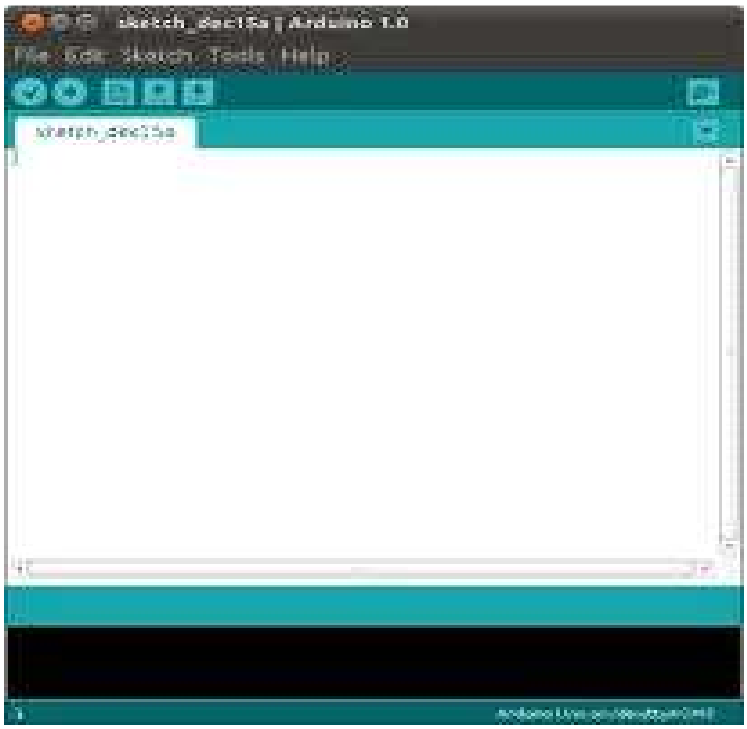


Figure 27- L'IDE Arduino

2. Le Terminal

Sur le logiciel X-CTU, nous n'avions pas pu bien observer les données qui étaient envoyées. La partie de la trame qui est en ASCII ne peut y être affichée correctement. D'où l'intérêt d'utiliser le logiciel Terminal. Ce logiciel nous permet de par contre de visualiser normalement toutes les données des capteurs que nous avons envoyés sur les modules Xbee. Ce logiciel est simple d'utilisation, il suffit tout simplement de brancher l'élément.

Mais pour recevoir ces données envoyées, il faut au préalable effectuer quelques paramétrages du logiciel.

Il faut branche le module sur le port auquel il est connecté, définir une vitesse de communication. Sur ce logiciel, il est possible de définir aussi la base dans

laquelle les données seront affichées. Nous avons entre autre, les bases en : ASCII, Hexadecimale, Binaire et Décimale.

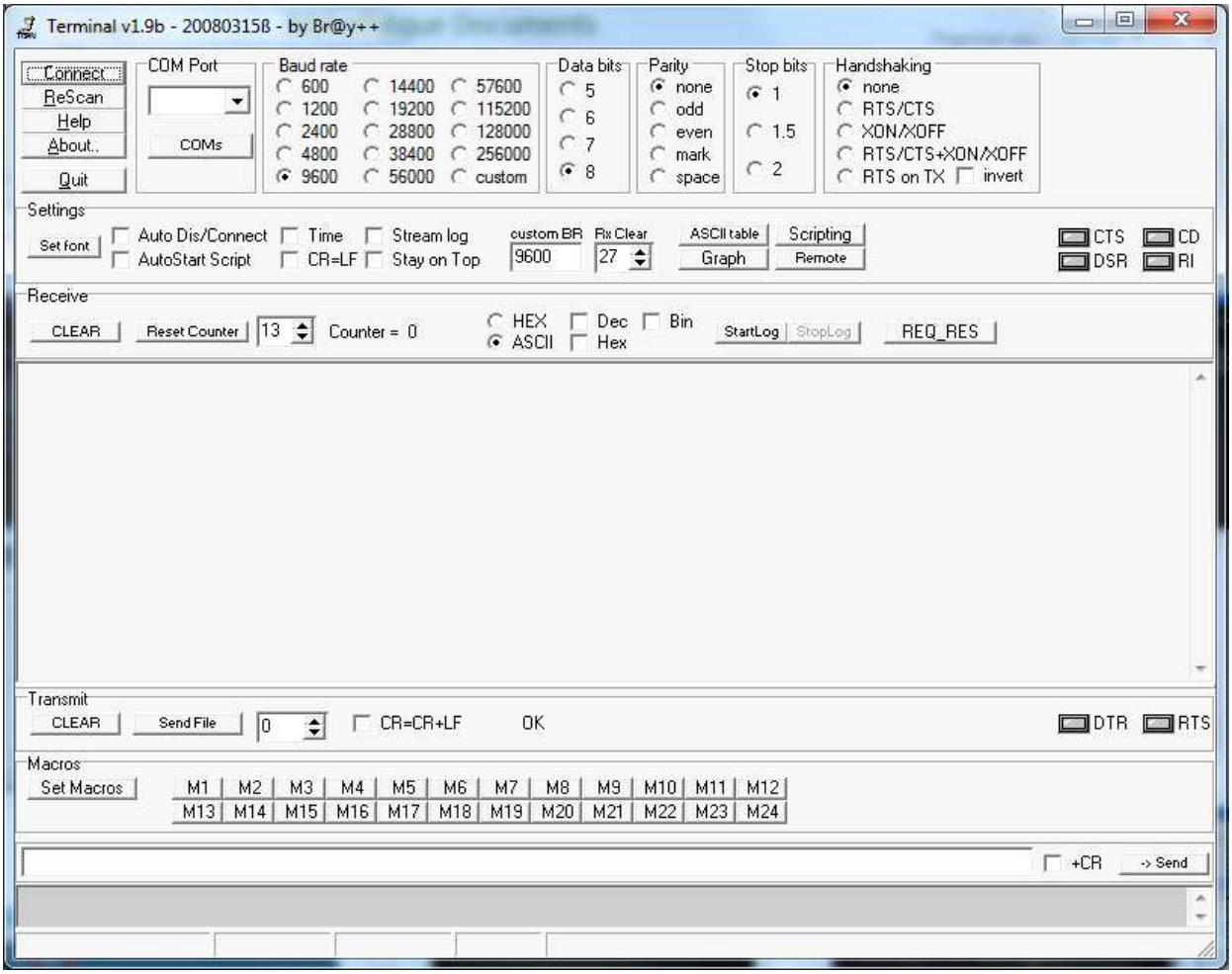


Figure 28- Le Terminal

G. Calendrier des tâches effectuées

L'image ci-dessous est une estimation des temps de réalisation des différentes tâches que nous avons réalisées durant notre Grand Projet.

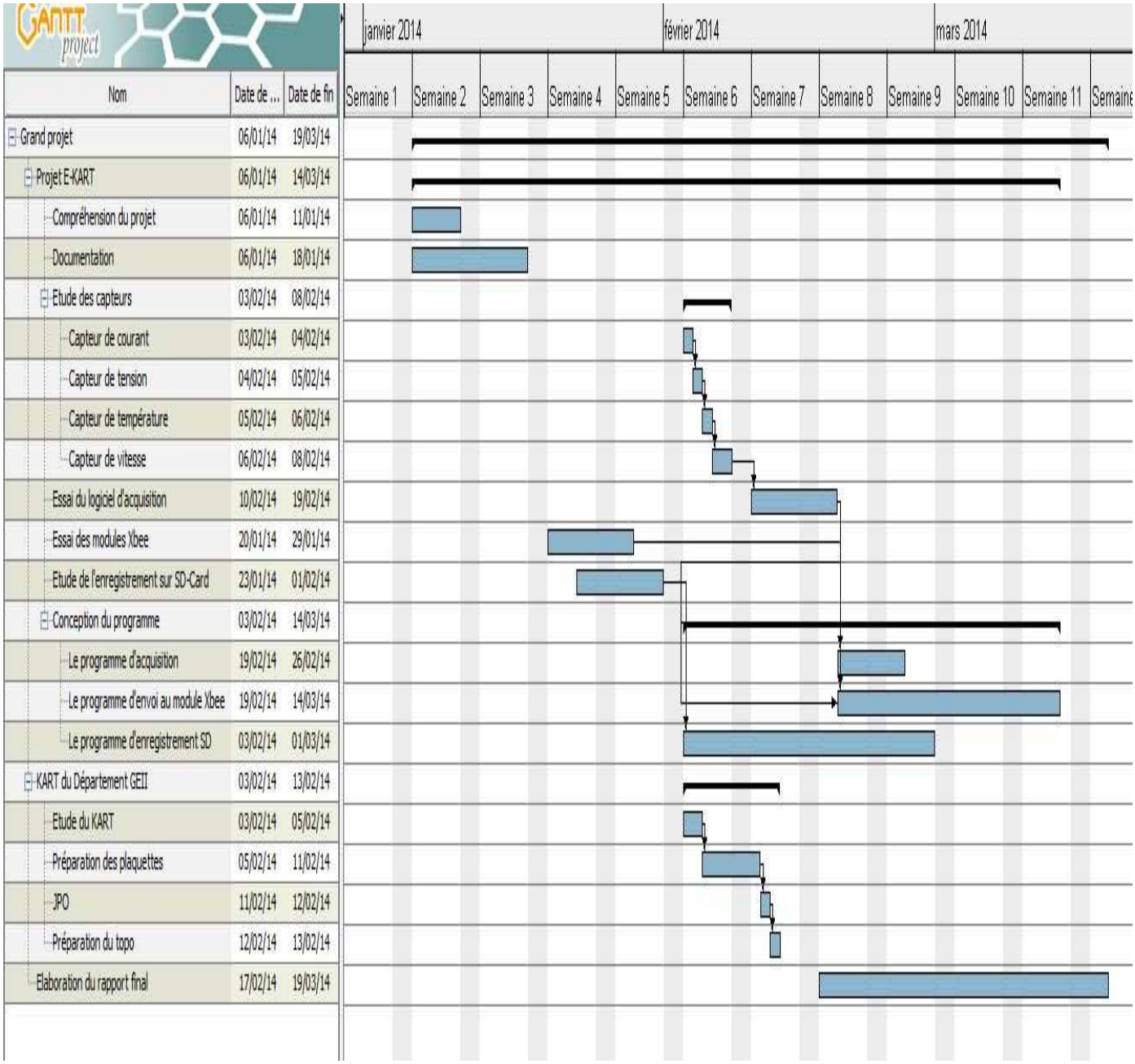


Figure 29 – Calendrier des tâches (Gantt Project)

Répartition des tâches

L'image ci-dessous met en relief, la répartition des différentes tâches que chacun de nous a pu réaliser :

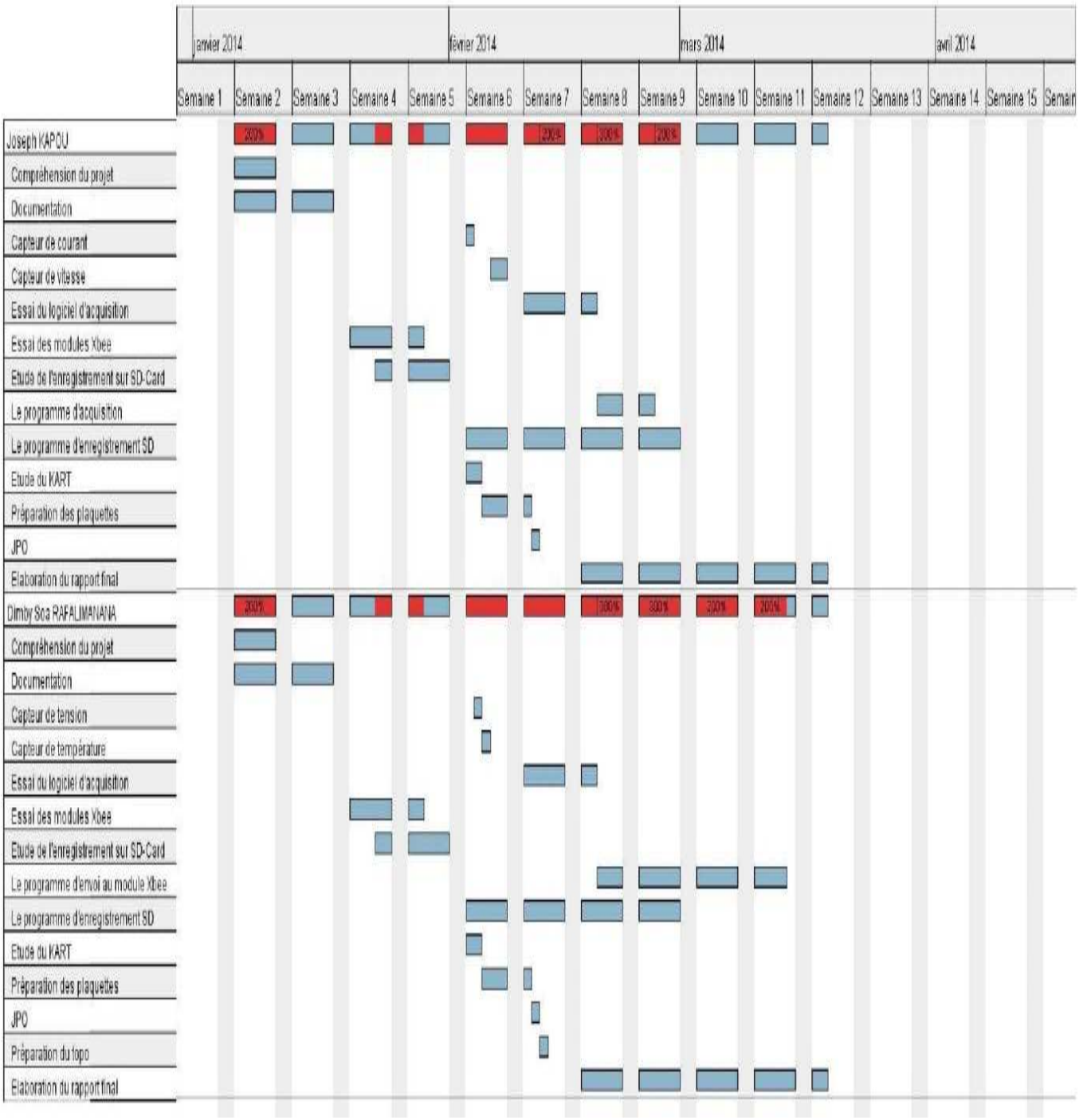


Figure 30 – Répartition des tâches (Gantt Project)

Chapitre II: Acquisition et traitement

A. Présentation et analyses

L'objectif de cette partie de notre programme est de recueillir les données analogiques ou numériques des capteurs du KART sur notre carte arduino DUE. Ces données seront traitées pour pouvoir constituer une trame qui sera envoyée par module Xbee au poste fixe au bord du circuit ainsi qu'au module d'affichage(pour le pilote) mais aussi sauvegardée sur la SD Card.

B. Diagramme de fonctionnement du programme

Notre système se divise en deux grandes parties : une partie gérée en interruption qui a pour rôle de recueillir les données et une autre partie qui est la tâche de fond qui s'occupe de l'envoi par module Xbee, l'envoi au module d'affichage ainsi que l'enregistrement sur la SD Card.

Les diagrammes suivants nous montrent le fonctionnement général de notre programme :

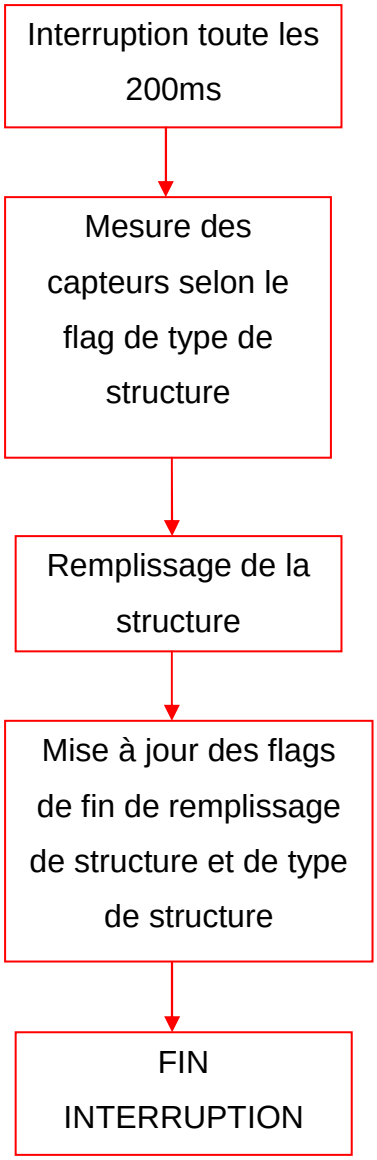


Figure 31 . Diagramme de la fonction acquisition gérée en interruption

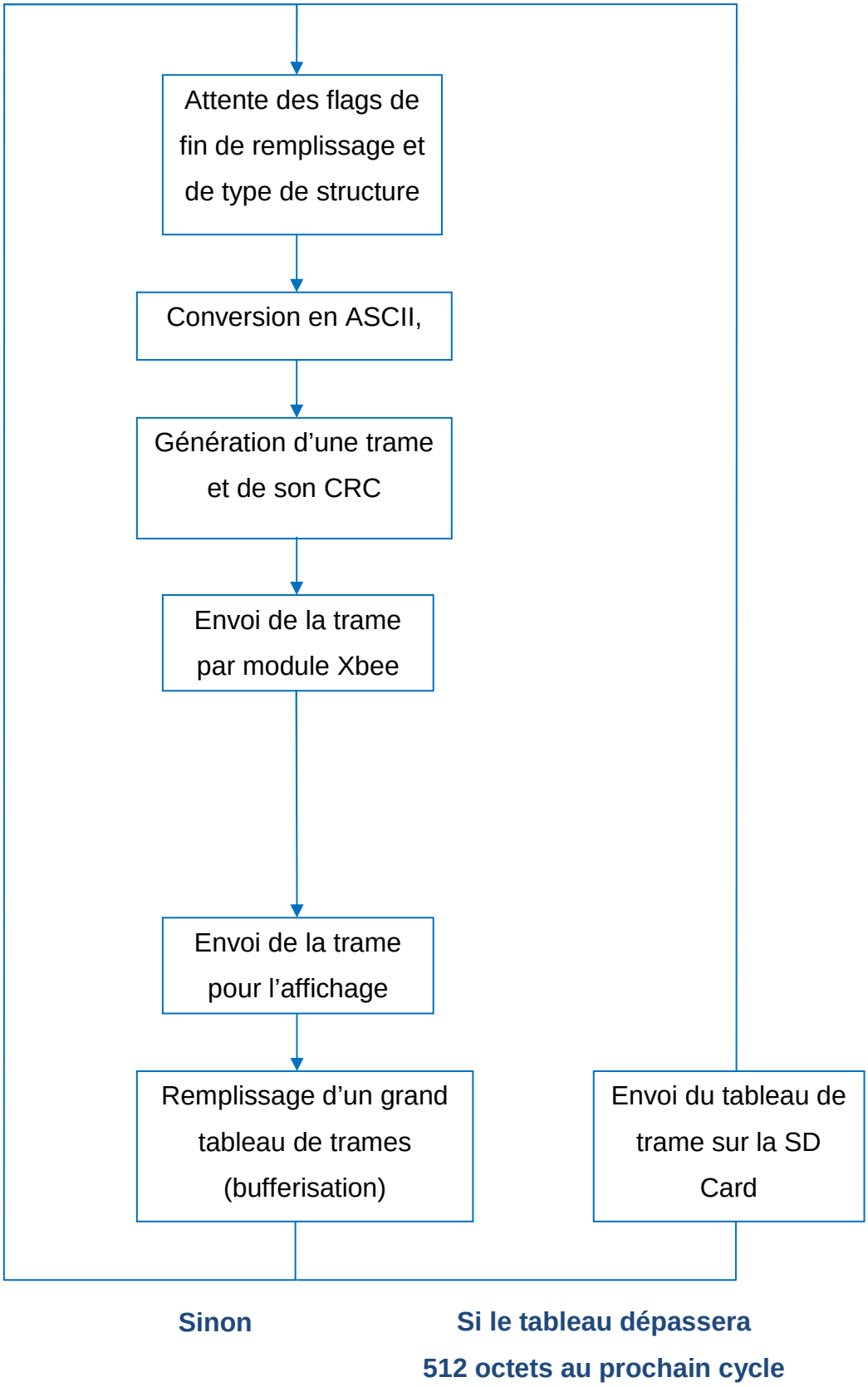


Figure 32 – Diagramme de fonctionnement des tâches de fond

C. L'acquisition et le traitement

1. Le rafraîchissement des données

Pour cadencer la mesure des capteurs, nous avons choisi de mettre en place dans notre programme une interruption toutes les 200 ms. Cette interruption appelle la fonction « TC3_Handler() ». Dans cette interruption, il est préférable de ne faire que quelques opérations ne nécessitant pas trop de temps. Les tâches lentes telles que l'envoi des trames Xbee vont être faites en scrutation dans le « loop() ».

a) Problématique

La Due est la dernière carte Arduino en date. Etant très différente des anciennes cartes Arduino, due notamment à son microcontrôleur SAM3X au lieu de microcontrôleur de type AVR comme les autres cartes Arduino. Ainsi, les ingénieurs de chez Arduino n'ont pas encore publié de références sur de nombreuses fonctionnalités, notamment les Timer Interrupts qui sont des interruptions internes.

b) Solution

Ainsi, il nous a fallu passer par les fonctions propres au microcontrôleur SAM3X pour générer une interruption.

Le SAM3X est équipé de 3 compteurs appelés TimerCounter (TC0, TC1, TC3) eux même constitués de 3 canaux sur lesquels peuvent être attachés des interruptions. Ces compteurs peuvent être incrémentés par un des horloges du microcontrôleur que nous pouvons voir dans le tableau ci-dessous.

Horloge	Fréquence
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5	Egale au MCK

MCK : Master Clock (horloge principale) de fréquence 84 MHz

Figure 33 - Les horloges du microcontroleur SAM3X

c) ***Configuration de l'interruption***

Pour configurer ces interruptions nous avons implémenté la fonction « startTimer ». Selon le compteur et le canal choisi, la fonction rattachée à l'interruption est comme suit :

Compteur (Counter)	Canal	Interruption (IRQ)	Fonction d'interruption
TC0	0	TC0_IRQn	TC0_Handler()
TC0	1	TC1_IRQn	TC1_Handler()
TC0	2	TC2_IRQn	TC2_Handler()
TC1	0	TC3_IRQn	TC3_Handler()
TC1	1	TC4_IRQn	TC4_Handler()
TC1	2	TC5_IRQn	TC5_Handler()
TC2	0	TC6_IRQn	TC6_Handler()
TC2	1	TC7_IRQn	TC7_Handler()
TC2	2	TC8_IRQn	TC8_Handler()

Figure 34 - Tableau récapitulatif de l'initialisation de l'interruption

Nous avons choisi le canal 0 du compteur TC1 pour notre interruption toutes les 200ms. Les fonctions NVIC (Nested Vector Interrupt Controller) permettent d'initialiser tous les registres du compteur ainsi que les registres d'interruption. L'initialisation d'une interruption tous les 200ms se présente alors comme suit :

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: Initialisation de l'interruption toutes les 200ms
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: Mars 2014
Le suivant permet d'initialiser une interruption toutes les 200ms en utilisant le timer Counter
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

voidsetup(){

/*Appel de la fonction d'interruption Start Timer
        -TC1:Timer Counter
        -0: Canal
        -TC3_IRQn: l'interruption
        -frequency: fréquence d'interruption en HZ (ici 5Hz)
        -L'interrupt Service Routine est le TC3_Handler
*/
        startTimer(TC1, 0, TC3_IRQn, 5);
```

```
}

voidloop(){
//ICI ON MET LES TACHES DE FOND//
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///// Fonction d'initialisation de l'interruption  ///////

voidstartTimer(Tc *tc, uint32_t channel, IRQn_Typeirq, uint32_t frequency) {

        pmc_set_writeprotect(false);  //désactivation de la protection de l'écriture des
           //registres

        pmc_enable_periph_clk((uint32_t)irq);      //Activation de l'horloge

  /*Configurations du compteur:
        - Mode waveform (TC_CMR_WAVE): signal du type PMW
         - Le compteur est remis à zéro par un trigger lorqu'il atteint la valeur RC
         (TC_CMR_WAVSEL_UP_RC)
        - Le compteur utilise le TIMER_CLOCK4
*/

    TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
    TC_CMR_TCCLKS_TIMER_CLOCK4);

    /*Initialisation de la fréquence du compteur*/
    uint32_trc = VARIANT_MCK/128/frequency; //128 parce que le TIMER_CLOCK4
                                                    //est choisi
    TC_SetRA(tc, 0,rc/2);
    TC_SetRC(tc, 0,rc);
    TC_Start(tc, 0);

    /*Initialisation de l'interruption  selon le canal choisi*/
    tc->TC_CHANNEL[channel].TC_IER=TC_IER_CPCS;
    tc->TC_CHANNEL[channel].TC_IDR=~TC_IER_CPCS;

    /*Autoriser les interruptions*/
    NVIC_SetPriority(irq, 1) ;                  //Mettre la priorité de l'interruption à 1 (0
    correspondant à la priorité la plus élevée et 16 la moins élevée)
    NVIC_EnableIRQ(irq);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Fonction d'interruption
//Cette fonction est appelée toutes les 200ms

void TC3_Handler()
{
     //Accepter l'interruption
    TC_GetStatus(TC1, 0);

    //ICI ON ECRIT LES OPERATIONS GEREES EN INTERRUPTION
```

```
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

2. Les mesures

La carte Arduino Due présente plusieurs broches d'entrée analogiques et numériques. Nous utilisons ces broches d'entrées pour gérer nos capteurs.

Une entrée analogique de la Due recueille la mesure d'un capteur (une tension entre 0 et 3,3 V) qui est représentée par une valeur entre 0 et 1023. Une entrée numérique détecte un niveau de tension soit 0, soit 1 (LOW ou HIGH).

Durant le projet, nous n'avons pas pu faire les tests des capteurs mais nous nous sommes référé aux documents des capteurs utilisés par les Master GSAT ISEE ayant travaillé sur le projet l'année dernière.

a) La mesure de la tension

D'après les données que nous avons sur les capteurs que nous avons vus dans le chapitre I, nous avons mis en place le programme suivant que nous avons intégré à notre programme final pour gérer le capteur de tension:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: Acquisition du niveau de batteries
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 11/03/2014
Le code qui suit permet de d'avoir en pourcentage le niveau de charge des batteries
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

float batterie;            //Variable globale permettant de stocker le niveau de batterie
                    //pour que la fonction mesure() puisse-y accéder
voidsetup() {
        Serial.begin(9600);  //Initialisation de la communication série
}

void loop() {

    floattension,tensionPRCENT;
    int a;

      a = analogRead(A0);         //stockage de la valeur récupéré dans la variable
                                  //« charge »
    tension = a*(3.3/1024);     //conversion de la valeur lue (binaire) en volt
    if(tension>4)                // si la tension est entre 4 et 5V cad ( entre 0 et 100%)
  {
        tensionPRCENT=(tension*100)/5;   // mettre sous forme de pourcentage
        batterie=(tensionPRCENT-80)*5;   //adaptation du cahier de charge 4V=0% et
```

```
                                                    //avoir un pourcentage en forme INTEGER
 }
else//si c'est inferieur 4V on affiche 0
 {
        tensionPRCENT=0;
        batterie=0;
 }
        Serial.println(batterie);       //affichage sur le moniteur pour le test
        delay(1000);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

b) ***La mesure de la température***

Il est à signaler que cette fonction n'est valable qu'entre les tensions 1,88V et 2,74V qui correspondent aux températures de 0 et 100℃.

Ci-après, nous avons le programme qui recueille la tension en sortie du capteur et qui en déduit la valeur de température correspondante.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: Acquisition de la température
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 11/03/2014
Le code qui suit permet de transformer la tension en sortie du capteur de
température etd'en déduire la température correspondante puis de l'afficher sur le
moniteur
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

floatval_temperature;               //variable globale pour stocker la valeur de
                                    //températureobtenue

voidsetup() {
        Serial.begin(9600);                 //Initialisation de l'affichage sur le moniteur
}

voidloop() {

        floatvolt,temp,vi;          //variables intermédiaires

        vi=analogRead(A0);      //stockage de la valeur récupérée dans la variable « vi »
        volt=(vi*3.3)/1024;     //règle de trois pour avoir la valeur de tension
                                //correspondante
if(volt>=1.88 && volt<2.74)       //si la tension est comprise dans la plage de
                                      //mesure
{
        temp=(volt-1.88);
        temp=(temp*10000)/86;       //calcul de la température à partir de la tension
```

```
        val_temperature=temp*1;       //avoir la valeur en forme entière
  }
Serial.println(val_temperature);    //Affichage du résultat sur le moniteur pour le test
delay(1000);
}
```

//

c) ***La mesure du courant***

Selon les données des capteurs que nous utilisons, nous avons la fonction suivante à programmer pour notre système :

De cette courbe, nous avons la fonction « tension(courant) » suivante:

$$f(x) = 0{,}00111x + 2{,}52264$$

$f(x): tension$

$x: courant$

Il nous reste alors à en déduire le courant selon la tension mesurée sur l'Arduino DUE. Nous pouvons voir ci-après le code qui nous a permis de faire cela.

//

```
/*PROJET E-KART
Sujet: Acquisition du niveau des batteries
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 11/03/2014
Le suivant permet de d'avoir le courant à partir de la valeur
de tension en sortie du capteur de courant et de l'afficher sur le moniteur
*/
```

//

```
Void setup() {
Serial.begin(9600);          //Initialiser le port serie du moniteur à 9600 bauds
}

Void loop() {

    //Déclaration des variables locales

    floatv_reference=2.52264;    //Tension de référence (lorsque Imesuré=0)
    floatcoeff=0.00111;          //Coefficient directeur
    intval_c;                    //valeur du courant en float
    float c;
    float volt;
    floatval_courant;            //Valeur de courant finale en entier à afficher

    c=analogRead(A0);            //lecture du port
    volt= (c*5)/1024;

    val_c=(volt - v_reference)/coeff;
    val_courant=val_c*1;
```

```
    Serial.println(val_courant);     //affichage sur le moniteur
    delay(1000);
}
```

//

d) ***La mesure de la vitesse***

Le capteur de vitesse qui est un capteur magnétique. Il délivre en sortie une impulsion chaque fois que la cible fixe à l'arbre de transmission passe. Il faut alors déterminer le temps entre 2 impulsions pour pouvoir déterminer la vitesse en tr/min.

La détection d'une impulsion est effectuée sur l'un des broches de l'Arduino grâce à une interruption liée à cette broche. Pour cela nous utilisons la fonction AttachInterrupt() disponible sur la Due. Cette interruption est laissée à sa priorité initiale 0 afin qu'elle soit de priorité supérieur à celle du Timer Interrupt.

Le programme suivant gère la détection des impulsions et calcule la vitesse correspondante :

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: Acquisition de la vitesse
Outil: Arduino
Réalisation: Dimby Soa RAFALIMANANA - Joseph KAPOU
Date: 15/03/2014
Le code qui suit permet d'afficher la vitesse du KART selon les impulsions simulées
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

volatile float temps = 0;          //Contient le temps entre 2 impulsions
volatile float time_last = 0;      //Mémorise le temps de la dernière impulsion
const int diametre=210;            //diamètre de la roue en mm

void setup()
{
 attachInterrupt(14, vit, FALLING); //interruption pour appeler la fonction "vit"
                                    //la prise d'interruption est sur la pin 14
                                    //l'interruption est sur front montant (RISING)
 Serial.begin(57600);

}

void loop()
{
  int rpm = 0;
  int vitesse;
  while(1)
  {
  if(time > 0)
```

```
    {
       rpm = 60*(1000000/(temps));                //calcul de rpm par rapport au temps
(tr/mn)
                                                  //entre deux interruptions
      vitesse= rpm * 3.1416 * diametre *0.00006;    //Calcul de la vitesse en km/h

     }
  Serial.print("rpm :");
  Serial.println(rpm);
  Serial.print("vitesse en km/h : ");
  Serial.println(vitesse);
  delay(1000);
   }
}

//Fonction appelée à chaque front descendant d'une impulsion
void vit()  //fonction d'interruption
{
   temps = (micros() - time_last);          // avoir le temps apres la derniere
                                            // prise de valeur de micros
   time_last = micros();                    //stocker le compteur micro dans la variable
                                             //de mémorisation

}
```

//

3. La mémorisation des données mesurées (remplissage de la structure)

Dans le but de faciliter le stockage des données et la constitution des trames à envoyer au module XBEE et au module d'affichage mais aussi à stocker dans la SD Card, nous avons mis en place une structure des données variables dans la trame. Nous pouvons voir dans la figure suivante la définition de la structure dans le programme.

```
//===============déclaration de la structure de trame============================================================
struct struct_trame {
uint8_t num_kart;          //Le numero du kart est sur un octet
uint32_t chrono;           //Le chrono est sur 24bits donc nous prenons 3 octets
uint32_t num_tram;         //Le numero de la trame en cours est sur 24bits donc nous prenons 3 octets
uint32_t temperature;      //La température varie de 0 à 999 dans la trame=> nous prenons 3 octets
uint32_t courant;          //Le courant varie de 0 à 999 dans la trame=> nous prenons 3 octets
uint8_t vitesse;           //La vitesse varie de 0 à 99 dans la trame=> nous prenons 3 octets
uint8_t batterie;          //La charge de la batterie varie de 0 à 99 dans la trame=> nous prenons 3 octets
};
//=============fin de déclaration de la structure=========================================================
```

Figure 35 – Image de capture de la déclaration de structure

Le remplissage de cette structure se fait après la mesure des capteurs.Vu que les trames sont différentes selon le nombre de capteurs mesurés, seules les données pertinentes correspondantes au type de trame seront récupérées de la structure.

Un flag de fin de remplissage est mis à jour pour spécifier que le remplissage est effectué.

4. Génération de la trame et la conversion en ASCII

a) *La trame*

La trame requise par le logiciel de réception au bord de la piste et le dispositif d'affichage est constituée d'une moitié en donnée Hexadécimale et d'une seconde moitié en ASCII. D'après ce qui a été fait l'année d'avant, la trame était toujours de même longueur vu que les capteurs étaient toujours rafraichis en même temps.

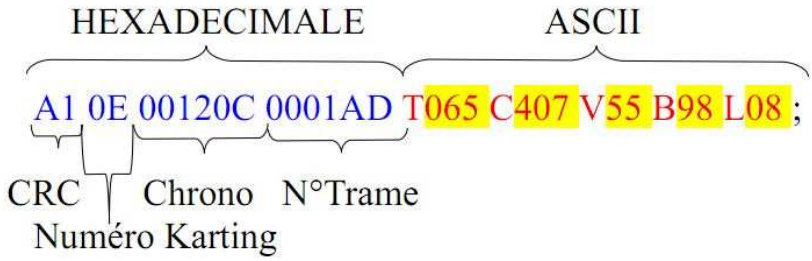


Figure 36 – Trame initiale

Cependant, après avoir optimisé le programme, nous avons 2 nouveaux types de trames. Selon le nombre de capteur mesuré, la trame peut être de 2 types :

- **A1 0E 00120C 0001AD T065 C407 V55 B98** lorsqu'on fait l'acquisition de 4 capteurs (toutes les secondes);
- **A1 0E 00120C 0001AD C407 V55** lorsqu'on fait l'acquisition de 2 capteurs (toutes les 200ms) ;

Ainsi, il nous faut d'abord faire une conversion en ASCII des données mesurées avant la constitution de la trame. C'est après la mise en forme de la trame que l'on calcule le checksum (CRC).

b) *Conversion en ASCII*

La fonction est effectuée par «conversion_ascii » dans notre programme.

La fonction sprintf() nous permet de faire la conversion en ASCII. La taille des données étant fixée à 2 ou 3 digits dans la trame, nous devons en prendre compte dans notre conversion. Par exemple, pour mettre la valeur de la température en

entier de la structure en ASCII dans un caractère nommé « température », la taille étant définie à 3 digits nous écrivons :

```
sprintf(temperature,"%03d",trame.temperature);
```

c) ***Génération d'une trame***

Après la conversion en ASCII, nous n'avons plus qu'à générer la trame. Pour cela, nous mettons toutes les valeurs de la trame dans un tableau de caractères que nommé « tab_tram[30]» dans notre programme. La taille du tableau n'est pas représentative de celle de la trame. Elle doit être seulement suffisamment grande pour pouvoir stocker la plus grande trame.

Pour la première partie en hexadécimal, le chrono et le numéro du kart sont de 3 octets (24bits) ce qui ne correspond à aucun type de variables déclarés en C++ qui sont de 8,16 et 32bits. Ainsi, dans la structure, nous avons déclaré ces données en variables de 32bits (voir Figure 35). A l'aide de masquages suivis de décalages nous affectons aux cases de « tab_tram» correspondantes les 24 bits 8 par 8.

```
tab_tram[0]=  trame.num_kart;
tab_tram[1]=(trame.chrono& 16711680)>>16;
tab_tram[2]=(trame.chrono& 65280)>>8;
tab_tram[3]=trame.chrono& 255;
```

Masquage et décalage pour avoir 1 octet dans une case de tab_tram

La constitution du reste de la trame se fait par une affectation simple de chaque variable à une case du tableau.

5. Calcul du CRC

Le CRC est une méthode utilisée pour la détection d'erreurs dans une trame envoyée entre un émetteur (notre système) et un récepteur (le logiciel d'acquisition). A chaque trame correspond un CRC. L'émetteur envoie avec la trame le CRC correspondant à celui-ci après l'avoir calculé. Le récepteur, à la réception de la trame recalcule le CRC et le compare à celui envoyé par l'émetteur. Lorsqu'il obtient le même CRC, la trame est correcte (voir Figure suivante).

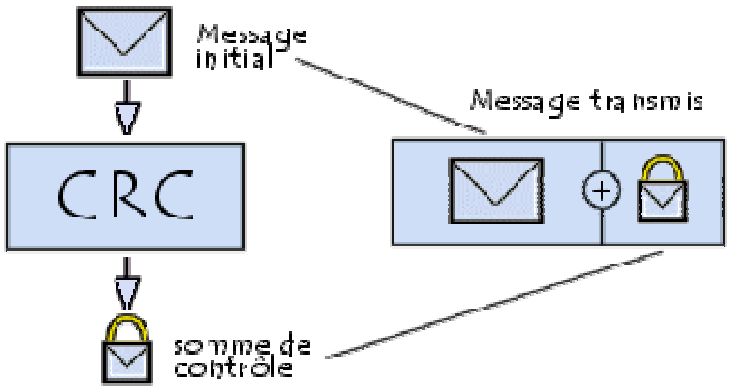


Figure 37 – Le CRC

Il existe plusieurs types de CRC qui ont leur propre méthode de calcul ainsi que son polynôme générateur.

Suite aux problèmes que nous avons concernant le logiciel récepteur trame que nous envoyons qui sont cités dans le chapitre suivant. Nous n'avons pas pu connaître avec exactitude la méthode qu'ils ont utilisée pour calculer le CRC ainsi que le polynôme générateur qu'ils ont utilisé. Nous savons pourtant que ces données doivent être similaires pour notre programme et leur logiciel afin que cela puisse fonctionner.

Sur le rapport de nos ainés, les étudiants en Master ISEE, nous voyons un CRC-16 tandis que dans la configuration du logiciel des étudiants en Informatique ce CRC semble être un CRC-8. Nous avons donc décidé de garder le CRC-16 que nos aînés ont mis en place dans leur programme.

Le sous-programme pour le calcul du CRC est alors comme suit :

```
//=======================fonction de calcul du crc-8 ==================
//Calcul la valeur du CRC d'un tableau de caractère et la renvoi

uint8_t Crc8(unsigned char *Adresse_tab , unsigned char Taille_max)
{
  uint8_t Polynome = 0xEA;                          //Polynome le plus utilisé pour le CRC-8
  unsigned char CptOctet = 0;
  unsigned char CptBit = 0;
  unsigned char Parity= 0;
  Crc = 0xFF;
  Polynome = 0xEA;
  for ( CptOctet= 0 ; CptOctet < Taille_max ; CptOctet++)
  {
    Crc ^= *( Adresse_tab + CptOctet);              //Ou exculsif entre octet message et CRC
    for ( CptBit = 0; CptBit <= 7 ; CptBit++)       /* Mise a 0 du compteur nombre de bits */
    {
      Parity= Crc;
      Crc >>= 1;                                    // Décalage a droite du crc
      if (Parity%2 == true ) Crc ^= Polynome;       // Test si nombre impair -> Apres decalage à droi
    }                                               // "ou exclusif" entre le CRC et le polynome gene
  }
  return(Crc);                                       // Renvoi le Crc(variable globale)
}
```

Chapitre III: Envoi de la trame par module Xbee et envoi au module d'affichage

A. Présentation et analyses

La trame préalablement constituée et stockée dans le tableau va être envoyée à un PC qui joue le rôle de poste fixe de surveillance au bord du circuit. A celui-ci est branché un module Xbee de réception de la trame. De plus, sur le PC est installé un logiciel qui permet d'afficher les valeurs mesurées.

De même, la trame est envoyée au module d'affichage pour que les mesures puissent être affichées sur un écran LCD et visibles par le pilote du KART.

B. Problèmes liés au logiciel d'acquisition

Le logiciel a été mis en œuvre par des étudiants en Informatique de l'IUT de l'année d'avant. Il a donc été conçu dans le cadre d'un projet tuteuré et posté sur forum e-Kart. Cependant, la version du logiciel que nous avons reçu n'était pas fonctionnelle. Il nous a alors fallu contacter les concepteurs pour leur demander une version qui marche avec les documents techniques le concernant. Nous n'avons pas eu de réponse de leur part jusqu'à la clôture du projet.

Le logiciel conçu par les étudiants en informatique est constitué de 2 parties :

- Le simulateur appelé Simu a pour rôle de simuler le Kart.

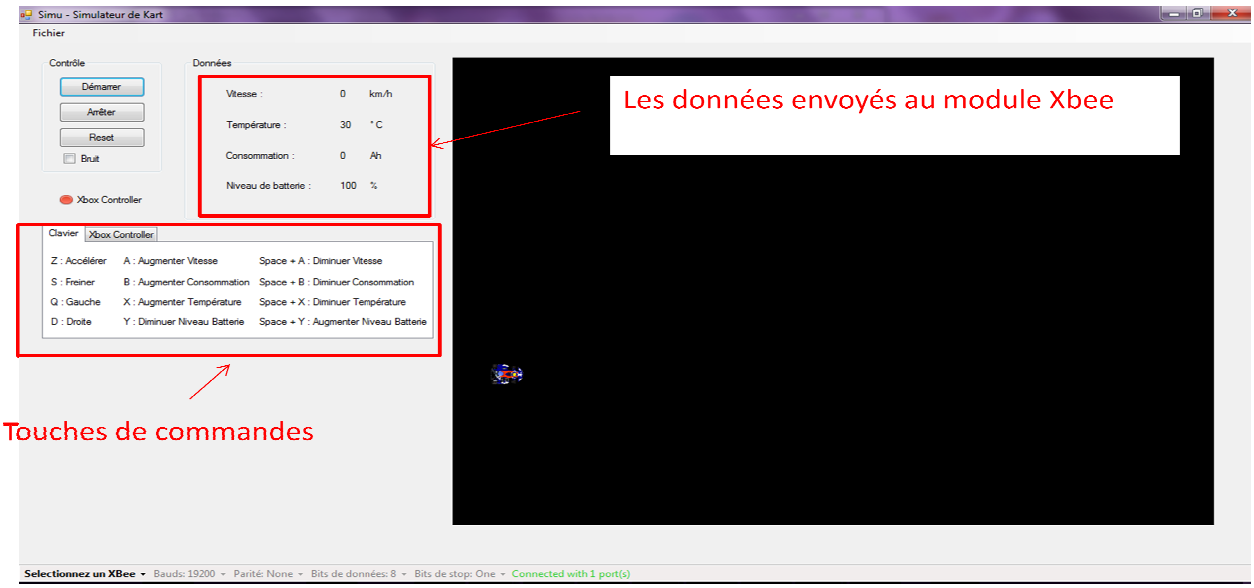


Figure 38 – Le logiciel Simu

Grâce aux touches du clavier ou une manette Xbox, on peut diriger le Kart virtuel et ainsi envoyer des données au module Xbee.

- Le logiciel d'acquisition proprement dit appelé CaptureServ : Il recueille les données venant du module Xbee branché au port USB du PC. Ces données seront ensuite stockées dans une base de données pour être récupérées par un Serveur Web. Les mesures peuvent ainsi être affichées sur le navigateur Internet du PC ou sur un Smartphone à système android. Cette partie est la plus importante pour notre projet. Cependant, c'est la partie qui ne fonctionne pas.

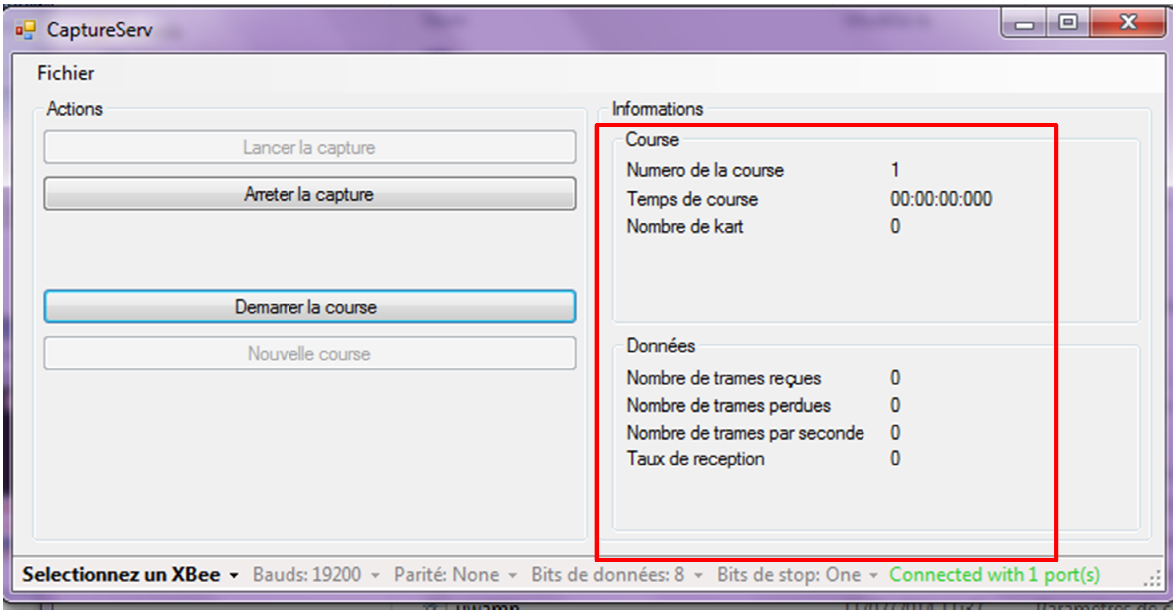


Figure 39 – L'outil CaptureServ du logiciel d'acquisition

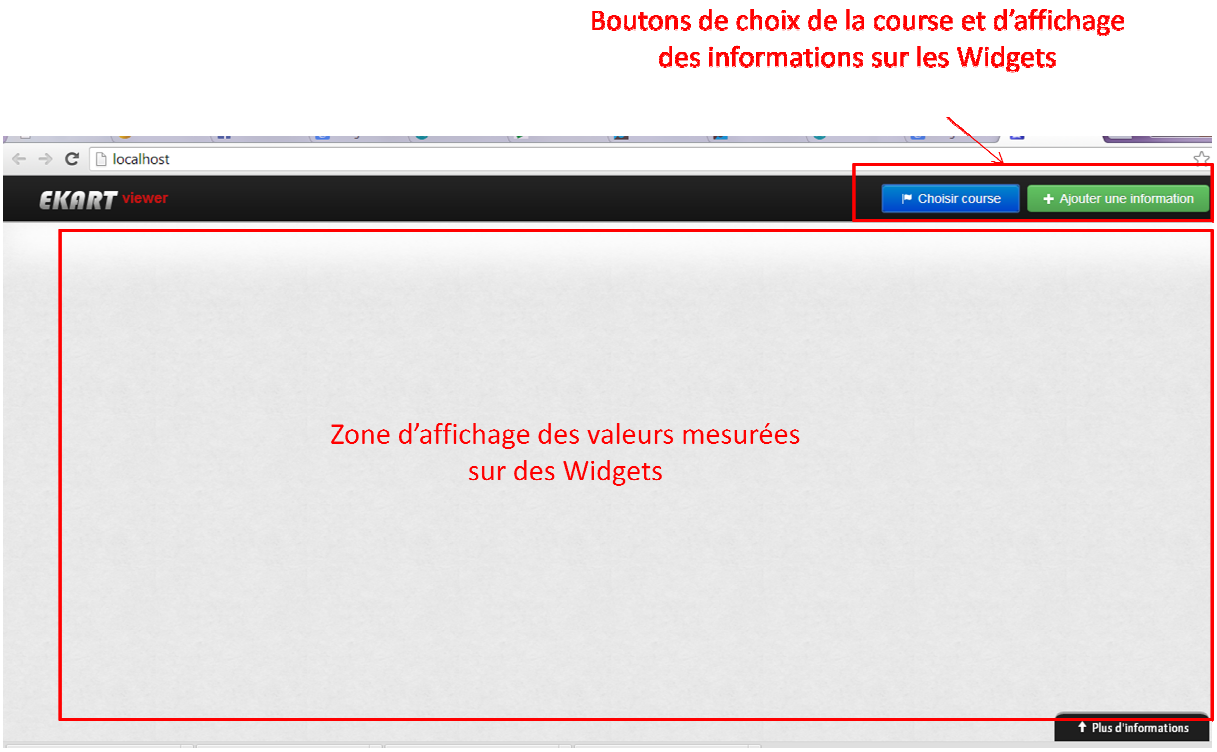


Figure 40 – Le navigateur lors de l'affichage de valeurs mesurées

C. Les contraintes d'envoi de la trame

Nous avons plusieurs contraintes à respecter pour l'envoi par module Xbee de la trame liées notamment au matériel que nous avons ainsi qu'au fonctionnement que nous voulons avoir.

Tout d'abord, les modules Xbee Pro que nous avons ont une portée Indoor de 90m et Outdoor de 1600m. Cela est largement suffisant pour l'utilisation que nous allons en faire.

De plus, sur les modules Xbee Pro dont nous disposons, nous pouvons avoir une vitesse de communication allant jusqu'à 115,2 Kbps(bit par seconde). Ainsi, avec la trame que nous avons à envoyer qui peut aller jusqu'à 22 octets (nous prendrons 25 octets). Le temps pour envoyer une trame est alors de :

$$Vitesse\ d'\ envoi\ trame = (8\ bits \times 25)/(115200\ bps)$$

$$Vitesse\ d'envoi\ trame = 1{,}7\ ms$$

D. Configuration des modules XBEE-Pro

Les modules XBEE-Pro sont configurés en point à point. Nous pouvons voir en annexe la configuration des modules Xbee-Pro.

E. L'envoi de la trame

Cette opération est effectuée par la fonction « envoi_xbee() de notre programme.

Grace à la fonction de génération de trame que nous avons vue précédemment, l'envoi de la trame par module Xbee est très simple. En initialisant le port Série utilisé par le module Xbee de la carte arduino Due à 115200 Bps, nous n'avons qu'à faire un Serial.write de chaque case du tableau et du Crc précédemment calculé (le Crc étant un unsigned int). Par exemple pour envoyer la trame correspondant la mesure de 2 capteurs seulement on a :

```
Serial.write(Crc);
for(int i=0;i<15;i++)
    {
Serial.write(tab_tram[i]) ;
}
```

Comme dans le tableau nous avons mis un octet dans une case, nous pouvons utiliser Serial.write pour envoyer la case au lieu de Serial.print.

Si l'on observe les trames envoyées sur le port Serie sur le Terminal, nous avons la figure qui suit :

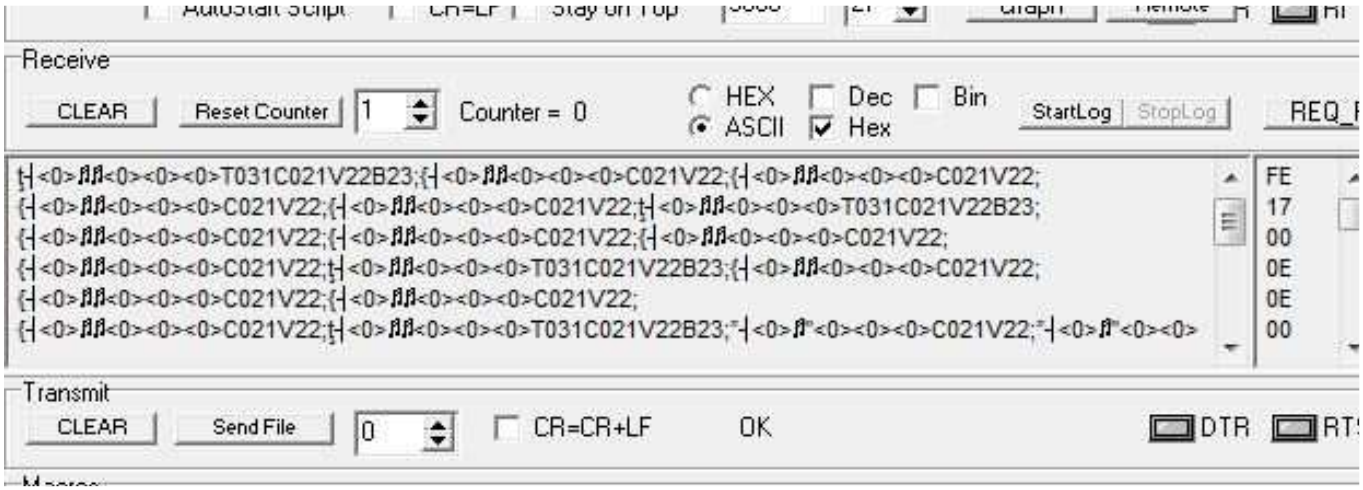


Figure 41 - Capture écran de trames envoyées sur le port Série du module Xbee

Comme la première partie de la trame est codée en hexadécimal tandis que le terminal est configuré en ASCII, nous ne pouvons pas distinguer clairement sur le Terminal. Grâce à la barre de droite nous pouvons voir ce que vaut chaque octet en Hexadécimal. Pour illustre nos propos, sur la figure que nous voyons **FE** représente le CRC en Hexa et **17** est le numéro du kart que nous avons mis dans le programme (équivalent à 23 en décimal).

F. Le sous-programme d'envoi de la trame par module XBEE

Nous avons sur le programme suivant, le code complet qui permet d'envoyer les trames XBEE dès leur mesure à leur envoi.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: Envoi des données par module Xbee
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 11/03/2014
La fonction suivante permet d'envoyer la trame générée par le module XBEE
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//============Fonction d'envoi des données du tableau par module
Xbee====================
voidenvoi_xbee()
{
   if(flag_acq==0 &&flag_fin_int==1)              //Si l'on a mesuré 4 capteurs
    {
   Serial.write(Crc);                          //Envoi du CRC

   for(int i=0;i<22;i++)
       {
   Serial.write(tab_tram[i]) ;    //Fait défiler les caractères du tableau et les envoi
       }
   flag_acq=1;   //Remet le flag du type de trame à 1 puisque
                                          //lors du prochain cycle on mesure 2 capteurs
   flag_mem_xbee=0;
    }

   elseif(flag_acq==1 &&flag_fin_int==1)          //Si l'on mesure 2 capteurs
    {
   Serial.write(Crc);                          //Envoi le CRC

   for(int i=0;i<15;i++)            //Fait défiler les caractères du tableau et les envoi
   {
       Serial.write(tab_tram[i]) ;
   }
}
```

//

G. Envoi de données au module d'affichage

Le module d'affichage est un système qui a été conçu par des étudiants en GEII de l'IUT dans le cadre de leur projet tuteuré. Ce module est constitué d'une carte Arduino UNO qui reçoit les données que nous envoyons et qui les affiche sur un afficheur pour que le pilote puisse voir en temps réelles l'état du KART.

Pour l'envoi des données nous utilisons une communication Série. Le fonctionnement de l'envoi est semblable à l'envoi sur le module Xbee mais sur un port Série différent

Suite à notre entretien avec les étudiants en GEII, il a été décidé que seule la partie en ASCII de la trame leur est nécessaire. Le sous-programme d'envoi au module d'affichage est alors comme suit :

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: Fonction d'envoi au module d'affichage
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 17/03/2014
La fonction suivante permet d'envoyer les données au module d'affichage
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

voidenvoi_affichage()
{

if(flag_acq==0 &&flag_fin_int==1)      //si on envoie la trame de  4 capteurs
{
for(int i=7;i<22;i++)
    {
Serial1.write(tab_tram[i]) ;
}
  }

elseif(flag_acq==1 &&flag_fin_int==1)    //si on envoie une trame de 2 capteurs
{
for(int i=7;i<15;i++)
    {
Serial1.write(tab_tram[i]) ;
}
}
```

//

Chapitre IV: La mémorisation sur la carte SD

A. Présentation et analyses

Dans cette partie, nous avons pour but de stocker les valeurs mesurés dans une SD Card. Pour cela nous utilisons la carte microshieldDEV-09802 de chez Sparkfunélectronics. Cette carte utilise le port SPI de l'Arduino pour envoyer des données à mémoriser sur une carte SD.

Au début du projet, nous avons enregistré les mesures en tableau en format .CSV qui peut s'ouvrir sur un tableur classique. Cependant, après que l'on a changé le délai de rafraîchissement de chaque capteur qui influe directement sur la longueur de la trame, cette méthode est alors devenue inintéressante. En plus du format, il a alors été décidé que nous allons mémoriser les trames elles même.

B. Les problématiques de la mémorisation sur SD Card

1. La SD Card

a) Les limites d'utilisation d'une SD Card

La mémorisation sur SD Card est une fonction très intéressante pour la technologie des cartes Arduino. Cependant, pour une utilisation comme la notre où l'enregistrement n'est pas continu, l'opération peut être délicate. En effet, pour pouvoir mémoriser dans la SD Card, il faut fermer le fichier.

Fermer le fichier chaque fois qu'une trame est générée et le rouvrir pour la mémorisation de la trame suivante peut être une solution. Cependant, cette solution est très couteuse en temps, vu qu'enregistrer une trame de 25 octets prend autant de temps que l'enregistrement d'un block entier. De plus, une SD Card a une limite du nombre d'enregistrement avant qu'elle ne tombe en panne. Ainsi il nous faut en tenir compte.

b) L'enregistrement d'un block (ou single block)

L'enregistrement de données sur une SD Card se fait block par block. Ainsi, le temps d'enregistrement minimum de données équivaut au temps d'enregistrement d'un block.

Pour la carte microshield SD de sparkfun, un block représente 512 octets de données et l'enregistrement de celui-ci est fait entre 850 microsecondes et 3 millisecondes. (source : biliothèqueSDFat/ Card performances).

Ainsi, pour avoir un fonctionnement optimal, nous essayerons d'accumuler les données en un block de 512 octets(ou un peu moins) et ensuite d'enregistrer le block sur la SD Card.

c) Le système de fichiers FAT 32

FAT32 (FAT pour *File Allocation Table ou* table d'allocation de fichiers) est un système de fichier Utilisant des adresses sur 28 bits, il permet de constituer des unités d'allocation de taille réduite sur des disques de taille importante. Ce système est surtout suggéré pour les mémoires de taille assez importantes. Puisque nous utilisons une SD Card d'environ 2Go, formater la SD Card en FAT32 est la solution optimale pour nous.

De plus, contrairement aux systèmes FAT16 et FAT, la FAT32 permet d'avoir des unités d'allocations (clusters) de 512 octets. Cela s'avère idéal pour nous car nous pouvons allouer un « block » d'enregistrement à un cluster de la SD Card.

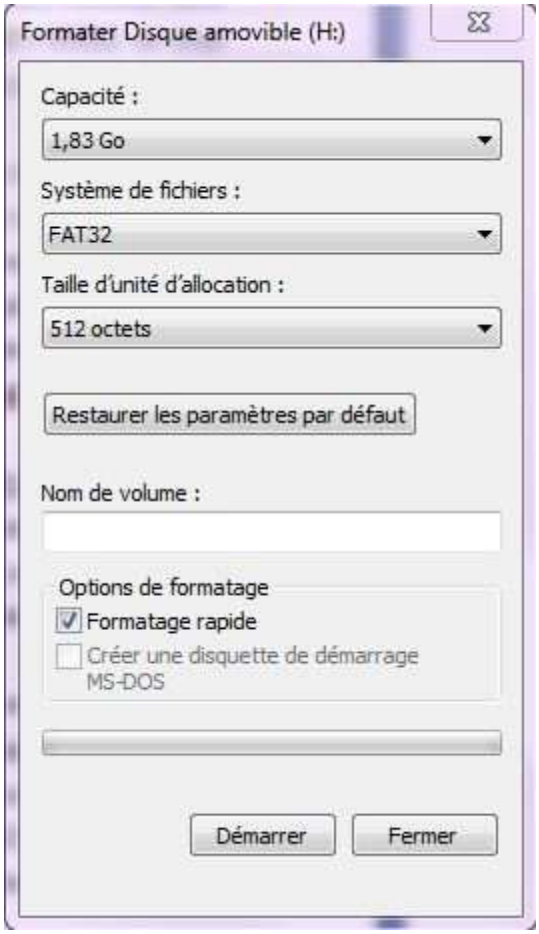


Figure 42 - Le système de fichier FAT 32 de la SD Card

Pour gérer l'enregistrement en FAT32, il est nécessaire d'installer une bibliothèque appelée <SDFat.h> avec la version la plus récente.

2. La carte microshield DEV-09802 de Sparkfun

La carte microshield a été fabriqué spécifiquement pour l'Arduino Uno et Mega. Or, sur ces cartes Arduino, les broches de communication SPI sont les broches 10 (SS), 11 (MOSI), 12 (MISO) et 13 (SCK). Sur la Due, par contre, les broches de communication SPI ne sont pas les même. Ainsi, il nous faudra ainsi modifier la carte microshield DEV-09802 pour qu'elle puisse fonctionner sur la Due en reliant les broches 10, 11, 12 et 13 de la carte microshield SD aux broches SPI de la DUE.

La figure suivante nous montre l'emplacement des broches SPI sur la UNO ainsi que sur la DUE.

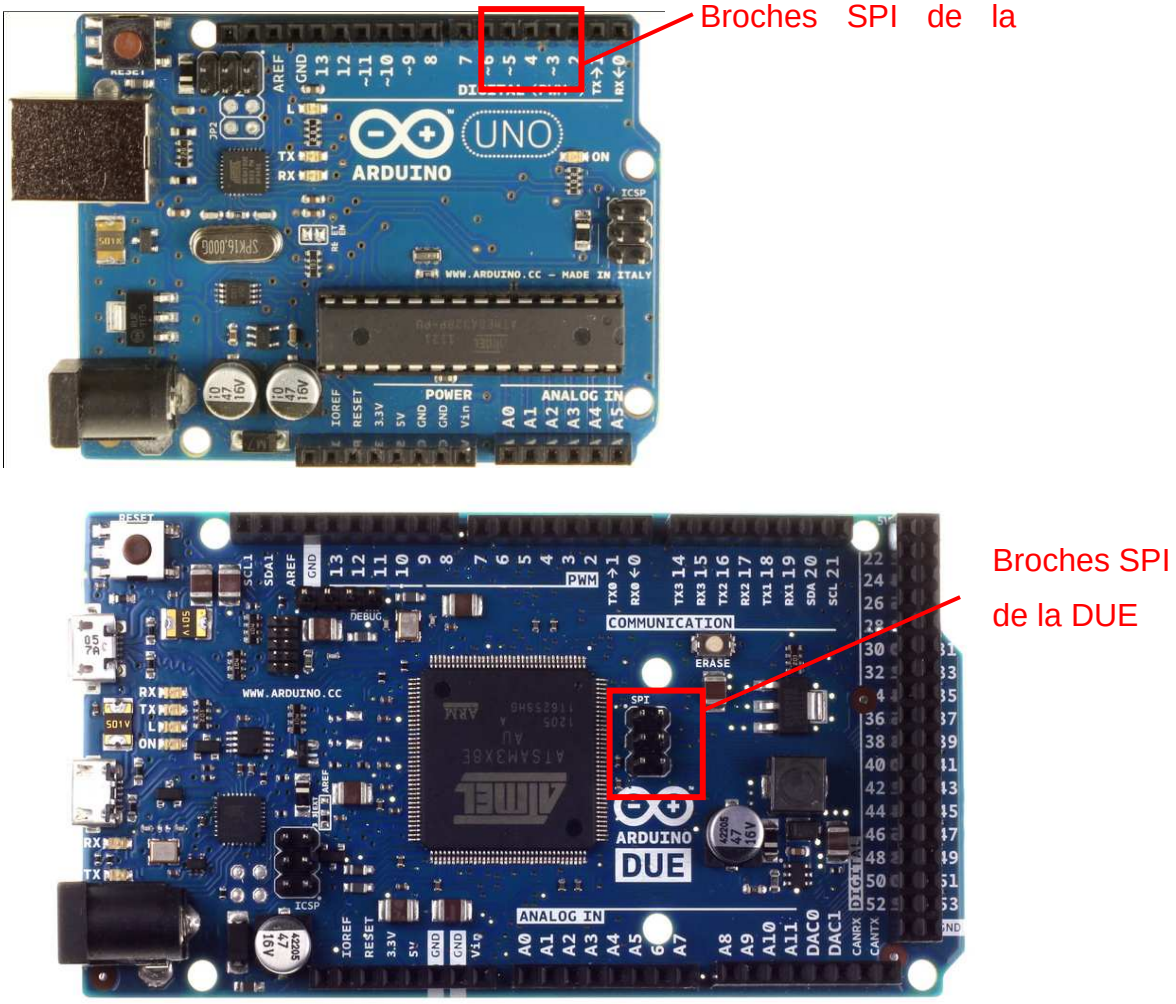


Figure 43 – Les broches SPI de la UNO et de la DUE

C. Mémorisation sur SD Card

La mémorisation sur SD Card utilise les fonctions liées à la bibliothèque SDFAT.h. Elle se déroule en 2 étapes. La première consiste à stocker les trames dans un buffer de manière à enregistrer le plus de donnée possible tout en ayant des données de moins de 512 octets. C'est à la deuxième qu'on envoi les données stockées dans le buffer.

a) Bufferisation

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: Bufferisation des données pour la sauvegarde sur SD Card
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 17/03/2014
La fonction suivante permet de stocker la trame dans un buffer avant
l'enregistrement sur la SD Card
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//==========Stockage des trames dans un grand tableau==================

voidstockage_supertrame_SD()
  {
if(flag_enregis==1)                //si on a une trame de 2 capteurs
        {
taille_buff+=sprintf(buff+taille_buff,"%c",(char)Crc);   //envoi du CRC dans le
                                                          //grand tableau

for(int j=0;j<7;j++)
        {
        taille_buff+=sprintf(buff+taille_buff,"%d",tab_tram[j]);    //envoi du numéro de
                                                                    // KART, du numéro du
                                                                   // chrono et du numéro
                                                                   // de trame dans le
                                                                   //grand tableau
        }
        taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[7]);  // enregistrement de "C"
                                                                  //dans le tableau
        decompo_envoi_SD(3,8);    //enregistrement des digits un par un de la valeur
                                   //courant
        taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[11]); // envoi le préfixe "V"
                                                                   //correspondant au type
                                                                   //de donnée "vitesse"

        decompo_envoi_SD(2,12);  //enregistrement des digits un par un de la valeur
                                   //courant
```

```
        taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[14]); // envoi ";" qui définit la
                                                                  //fin de la trame
}

elseif(flag_enregis==0 )
      {

taille_buff+=sprintf(buff+taille_buff,"%c",(char)Crc);      //envoi du CRC dans le grand
tableau

for(int i=0;i<7;i++)
 {
taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[i]);      //envoi du numero de KART,
du numero du chrono et du numero de trame dans le grand tableau
 }

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[7]);     // enregistrement de "T" dans
le tableau

decompo_envoi_SD(3,8);                              //envoi un par un des digits des
valeurs de température

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[11]); // envoi du prefixe "C"

decompo_envoi_SD(3,12);

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[15]); // envoi du prefixe "v"

decompo_envoi_SD(2,16);

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[18]); // envoi du prefixe "B"

decompo_envoi_SD(2,19);

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[21]); // envoi ";" qui definit la fon de
la trame

      }
 }
```

//

b) ***Enregistrement sur SD Card***

//

```
/*PROJET E-KART
Sujet: sauvegarde sur SD Card
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 17/03/2014
```

La fonction suivante permet d'enregistrer les trames stockées dans la SD Card
*/
//
//=============fonction d'enregistrement dans la SD card==================

```
voidenregis_SD()
{
if(flag_SD==1)           //Si le fichier a été fermé
  {
        file.open(&root, nom, O_CREAT | O_APPEND | O_WRITE);   //ouvrir le fichier
                                //si existant et écrire à la fin ou créer un fichier et l'ouvrir
        flag_SD=0;
  }
if((taille_buff>489 &&flag_acq==0)|(taille_buff>497 &&flag_acq==1))       //Si le
                                        // block de 512 octets sera rempli au prochain cycle
  {
    file.write(buff);    //écrire les trames du grand tableau dans la SD CARD
    file.close();        //fermer le fichier pour finaliser l'enregistrement
    flag_SD=1;           //mettre à jour le flag du fin d'enregistrement
    taille_buff=0;
  }
}
```

//

¨PROGRAMME COMPLET

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*PROJET E-KART
Sujet: PROGRAMME E-KART FINAL
Outil: Arduino
Réalisation: DimbySoa RAFALIMANANA - Joseph KAPOU
Date: 17/03/2014
Le programme suivant permet de
*/
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include <ArduinoStream.h>
#include <bufstream.h>
#include <ios.h>
#include <iostream.h>
#include <istream.h>
#include <MinimumSerial.h>
#include <ostream.h>
#include <Sd2Card.h>
#include <SdBaseFile.h>
#include <SdFat.h>
#include<SdFatConfig.h>
#include<SdFatmainpage.h>                                    //Déclaration de la bibliothèque
SDFat.h et tous ses composants
#include <SdFatStructs.h>
#include <SdFatUtil.h>
#include <SdFile.h>
#include <SdInfo.h>
#include <SdSpi.h>
#include <SdStream.h>
#include <SdVolume.h>

#include <SPI.h>


//==============déclaration des structure de la trame==============
structstruct_trame {
uint8_t num_kart;
uint32_t chrono;
uint32_t num_tram;
uint32_ttemperature;
uint32_t courant;
uint8_t vitesse;
uint8_t batterie;
};

//==============déclaration des tableaux de simulation=============
volatilestruct_trame trame;
char temperature[3];
char courant[3];
char vitesse[2];
char batterie[2];

//==============fin de déclaration de la structure=================
uint8_t numero_tram;
volatileintval_courant;
volatileintval_tension;          //Déclaration des variables globales utilisées par les fonctions de
mesures
intval_temperature;
intval_vitesse;
```

```
uint8_t cpt=0;

//inttab_temperature[5]={10,28,31,6,96};
//inttab_courant[5]={100,29,407,301,350};
inttab_vitesse[5]={0,20,30,40,50};
//inttab_tension[5]={98,96,47,30,0};

int flag=0;                 //flag si les données sont à jour ou pas
intflag_acq=0;              // flag pour le choix des données à mesurer; 0:4 capteurs et 1 si 2 capteurs
intflag_SD=1;
intflag_mem_trame=0;
intflag_fin_int=0;
intflag_enregis=0;


uint8_t Crc=0;                                  // pour verifier la trame et mesurer sa longueur
unsigned char tab_tram[25];                     // tableau permettant de stocker les valeurs de la trame
pour calculer le Crc
//char buff_SD[512];
char buff[700];

longlongtaille_buff;
//uint32_t buff_var;

char nom[] = "bim.txt";//Create an array that contains the name                    of our file.

Sd2Card card;
SdVolume volume;
SdFile root;
SdFile file;
/*********************************Fonction Setup*********************************************/
//C'est ici que l'on initialise tous les paramètres
voidsetup() {
//intitialisation de l'interruption toutes les 200ms: 5Hz
startTimer(TC1, 0, TC3_IRQn, 5);    //Utiliser le TimerCounter 1, Canal 0, Interruption TC3_IRQn, 5Hz

//Initialisation de l'envoi Xbee
Serial.begin(57600);
//Initialisation de la communication série au module d'affichage
Serial1.begin(57600);
//Initialisation de la SD Card
pinMode(10, OUTPUT);              //pin 10 Chip Select de la microshield SD Card
card.init(SPI_EIGHTH_SPEED);                    //Initialiser la SD Card
volume.init(&card);             //Initialiser un volume sur la SD Card
root.openRoot(&volume);         //Choisir le volume comme place d'ouverture du fichier
}
/*****************************Fin des initialisations*************************************/

/*****************************Fonction loop*********************************************/
//C'est ici que se passe les tâches de fond
void loop() {
conversion_ascii();               //fonction de conversion en ASCII
generation_trame();               //fonction de generation de trame
envoi_xbee();                   //fonction d'envoi xbee
envoi_affichage();                //fonction d'envoi au module d'affichage
increment_cpt();                  //incrementation du compteur de simulation des capteurs
stockage_supertrame_SD();         //Stockage dans tableau de trame (bufferisation)
enregis_SD();                   //Enregistrement sur la SD Card
}
/***************************Fin de la fonction loop*************************************/
```

```
/*************************fonction acquisition gérée en interruption**********************/
//C'est la fonction déclenchée toutes les 200ms
void TC3_Handler()
{
TC_GetStatus(TC1, 0);           //déclencher l'interruption

mesure();                       //mesure des capteurs
remplissage_structure();        //remplissage de la structure
}


//*************************fonction mesure*************************************************/
voidmesure() {

if (flag_acq==0)                //si on mesure 4 capteurs
     {
mesure_vitesse();
mesure_courant();
mesure_tension();
mesure_temperature();
     }

elseif(flag_acq==1)             //si on mesure 2 capteurs
      {
mesure_vitesse();
mesure_courant();
      }
}
/*****************************fin de la mesure*********************************************/

//*************************fonction remplissage structure**********************************/
//C'est cette fonction qui recueille les mesures et les met dans une structure
voidremplissage_structure() {
if (flag_acq==0)                        //si on mesure 4 capteurs
     {
trame.num_kart=23;                      //Le numero du kart est fixe

trame.chrono=millis();                  //
trame.num_tram=numero_tram;             //stocke le numero de trame dans la sturcture
trame.temperature=val_temperature;      //Toutes les données sont mises à jour
trame.courant=val_courant;              //
trame.vitesse=val_vitesse;              //
trame.batterie=val_tension;             //

flag_fin_int=1;                 //flag qui marque que la fin de l'interruption
                                //La mise à jour est effectuée
     }

elseif(flag_acq==1)                     //si on mesure 4 capteurs
      {
trame.num_kart=23;                      //Le numero du kart est fixe

trame.chrono=millis();                  //
trame.num_tram=numero_tram;             //Seul le courant et la vitesse sont mis à jour
trame.courant=val_courant;              //
trame.vitesse=val_vitesse;              //

flag_fin_int=1;                 //flag qui marque que la fin de l'interruption
```

```
                                //La mise à jour est effectuée
        }
    }

/**********************Les tâches de fonds*******************************/
//=========================Conversion ascii==========================
Void conversion_ascii()
{
if(flag_acq==0 &&flag_fin_int==1)          //si on mesure 4 capteurs et les données sont à jours
{
sprintf(temperature,"%03d",trame.temperature);
sprintf(courant,"%03d",trame.courant);
sprintf(vitesse,"%02d",trame.vitesse);
sprintf(batterie,"%02d",trame.batterie);
  }
elseif(flag_acq==1 &&flag_fin_int==1)    //si on mesure 2 capteurs et les données sont à jours
{
sprintf(courant,"%03d",trame.courant);
sprintf(vitesse,"%02d",trame.vitesse);
}
}
//====================génération de la trame=========================

voidgeneration_trame()
{
if(flag_acq==0 &&flag_fin_int==1)      //si on mesure 4 capteurs
  {
tab_tram[0]=trame.num_kart;
tab_tram[1]=(trame.chrono& 16711680)>>16;
tab_tram[2]=(trame.chrono& 65280)>>8;
tab_tram[3]=trame.chrono& 255;
tab_tram[4]=(trame.num_tram& 16711680)>>16;
tab_tram[5]=(trame.num_tram& 65280)>>8;
tab_tram[6]=trame.num_tram& 255;

tab_tram[7]='T';
tab_tram[8]=temperature[0];
tab_tram[9]=temperature[1];
tab_tram[10]=temperature[2];

tab_tram[11]='C';
tab_tram[12]=courant[0];
tab_tram[13]=courant[1];
tab_tram[14]=courant[2];

tab_tram[15]='V';
tab_tram[16]=vitesse[0];
tab_tram[17]=vitesse[1];


tab_tram[18]='B';
tab_tram[19]=batterie[0];
tab_tram[20]=batterie[1];
tab_tram[21]=';';


Crc8(tab_tram,23);
  }

elseif(flag_acq==1 &&flag_fin_int==1)      //si on mesure 2 capteurs
```

```
 {
tab_tram[0]=trame.num_kart;
tab_tram[1]=(trame.chrono& 16711680)>>16;
tab_tram[2]=(trame.chrono& 65280)>>8;
tab_tram[3]=trame.chrono& 255;
tab_tram[4]=(trame.num_tram& 16711680)>>16;
tab_tram[5]=(trame.num_tram& 65280)>>8;
tab_tram[6]=trame.num_tram& 255;

tab_tram[7]='C';
tab_tram[8]=courant[0];
tab_tram[9]=courant[1];
tab_tram[10]=courant[2];

tab_tram[11]='V';
tab_tram[12]=vitesse[0];
tab_tram[13]=vitesse[1];
tab_tram[14]=';';

Crc8(tab_tram,15);
}
}


/**********************fonction d'envoi de la trame par module xbee*********************/
Void envoi_xbee()
{
numero_tram++;
if(flag_acq==0 &&flag_fin_int==1)      //si on envoie la trame de  4 capteurs
{

Serial.write(Crc);
for(int i=0;i<22;i++)
    {
Serial.write(tab_tram[i]) ;
}
 }

elseif(flag_acq==1 &&flag_fin_int==1)    //si on envoie une trame de 2 capteurs
{
Serial.write(Crc);
for(int i=0;i<15;i++)
    {
Serial.write(tab_tram[i]) ;
}
 }
}

/***************fonction d'envoi de la trame au module d'affichage**********************/
voidenvoi_affichage()
{
if(flag_acq==0 &&flag_fin_int==1)      //si on envoie la trame de  4 capteurs
{
for(int i=7;i<22;i++)
    {
Serial1.write(tab_tram[i]) ;
}
    flag_acq=1;
    flag_enregis=0;
 }
```

```
elseif(flag_acq==1 &&flag_fin_int==1)    //si on envoie une trame de 2 capteurs
{
for(int i=7;i<15;i++)
    {
Serial1.write(tab_tram[i]) ;
}
flag_enregis=1;

//mise à jour du flag de choix du nombre de capteurs à mesurer
if(flag_mem_trame==3)
     {
flag_acq=0;
flag_mem_trame=0;
    }
else{flag_mem_trame+=1;}
}
}

//================Stockage des trames dans un grand tableau==============

voidstockage_supertrame_SD()
 {
if(flag_enregis==1)                //si on a une trame de 2 capteurs
        {

taille_buff+=sprintf(buff+taille_buff,"%c",(char)Crc);      //envoi du CRC dans le grand tableau

for(int j=0;j<7;j++)
 {
taille_buff+=sprintf(buff+taille_buff,"%d",tab_tram[j]);    //envoi du numero de KART, du numero du
chrono et du numero de trame dans le grand tableau
 }

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[7]);  // enregistrement de "C" dans le tableau

decompo_envoi_SD(3,8);    //enregistrement des digits un par un de la valeur courant

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[11]); // envoi le prefixe "V" correspondant au type de
donnee "vitesse"

decompo_envoi_SD(2,12);  //enregistrement des digits un par un de la valeur courant

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[14]); // envoi ";" qui definit la fin de la trame
 }

elseif(flag_enregis==0 )
       {

taille_buff+=sprintf(buff+taille_buff,"%c",(char)Crc);      //envoi du CRC dans le grand tableau

for(int i=0;i<7;i++)
 {
taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[i]);      //envoi du numero de KART, du numero du
chrono et du numero de trame dans le grand tableau
 }

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[7]);     // enregistrement de "T" dans le tableau

decompo_envoi_SD(3,8);      //envoi un par un des digits des valeurs de température

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[11]); // envoi du prefixe "C"
```

```
decompo_envoi_SD(3,12);

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[15]); // envoi du prefixe "v"

decompo_envoi_SD(2,16);

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[18]); // envoi du prefixe "B"

decompo_envoi_SD(2,19);

taille_buff+=sprintf(buff+taille_buff,"%c",tab_tram[21]); // envoi ";" qui definit la fon de la trame

      }
 }


//====================fonction d'enregistrement dans la SD card=====================

Void enregis_SD()
{

if(flag_SD==1)          //Si le fichier a été fermé
 {
        file.open(&root, nom, O_CREAT | O_APPEND | O_WRITE);    //ouvrir le fichier si existant et
                                                                 //écrire à la fin ou créer un fichier et l'ouvrir
        flag_SD=0;
 }

if((taille_buff>489 &&flag_acq==0)|(taille_buff>497 &&flag_acq==1))       //Si le block de 512 octets
                                                                          //sera rempli au prochain cycle
 {
file.write(buff);    //écrire les trames du grand tableau dans la SD CARD
file.close();        //fermer le fichier pour finaliser l'enregistrement
flag_SD=1;           //mettre à jour le flag du fin d'enregistrement
taille_buff=0;
 }
}
//================================================================================

  //==================decomposition d'envoi
sd=========================================
   //cette fonction permet d'envoyer un par un les digits des valeurs mesurés de la partie en ASCII de
la trame
Void decompo_envoi_SD(intlongueur,int i)
{
if (longueur==2)
 {
taille_buff+=sprintf(buff+taille_buff,"%c",(char)tab_tram[i]);      // copier le premier digit dans le
                                                                    //tableau de trame
taille_buff+=sprintf(buff+taille_buff,"%c",(char)tab_tram[i+1]);    // copier le second digit dans le
                                                                    //tableau de trame
 }

else if (longueur==3)
   {
taille_buff+=sprintf(buff+taille_buff,"%c",(char)tab_tram[i]);      // copier le premier digit dans le
                                                                    // tableau de trame
taille_buff+=sprintf(buff+taille_buff,"%c",(char)tab_tram[i+1]);    // copier le second digit dans le
                                                                    // tableau de trame
taille_buff+=sprintf(buff+taille_buff,"%c",(char)tab_tram[i+2]);    // copier le troisième digit dans le
```

```
                                                                    //tableau de trame
   }
}
//====================fin décomposition d'envoi sd==============================

//======================fonction de calcul du crc================================

uint8_t Crc8(unsigned char *Adresse_tab , unsigned char Taille_max)
{
    uint8_tPolynome = 0xEA;
    unsigned char CptOctet = 0;
    unsigned char CptBit = 0;
    unsigned char Parity= 0;
    Crc = 0xFF;
    Polynome = 0xEA;                          // ancien Polynôme = 2^15 + 2^13 + 2^0 = 0xA001.
 for ( CptOctet= 0 ; CptOctet<Taille_max ; CptOctet++)
 {
Crc ^= *( Adresse_tab + CptOctet);              //Ou exculsif entre octet message et CRC
   for ( CptBit = 0; CptBit<= 7 ; CptBit++)       /* Mise a 0 du compteur nombre de bits */
   {
Parity= Crc;
Crc>>= 1;                                 // Décalage a droite du crc
     if (Parity%2 == true )Crc ^= Polynome;        // Test si nombre impair -> Apres decalage à droite il
                                                    //y aura une retenue
   }                                      // "ou exclusif" entre le CRC et le polynomegenerateur.
 }
return(Crc);                                  // Recupere le Crc
}

//Fonction de RAZ du compteur utilisé pour la simulation de la vitesse

Void increment_cpt()
{
    if (cpt==4)
    {
          cpt=0;
    }
    else
    {
           cpt+=1;
    }
}

//====================================fonction mesure vitesse==================
//affectation de la valeur du tableau

Void mesure_vitesse()
{
val_vitesse=tab_vitesse[cpt];                 //vitesse simulée à l'aide du tableau
}
//================================fonction de mesure du courant================
intmesure_courant()
{
      floatv_reference=2.52264;    //Tension de référence (lorsque Imesuré=0)
      floatcoeff=0.00111;          //Coefficient directeur
      intval_c;                   //valeur du courant en float
      float c;
      float volt;
```

```
      c=analogRead(A9);          //lecture du port
      volt= (c*3.3)/1024;

      val_c=(volt - v_reference)/coeff;
      val_courant=val_c*1;
      return (val_courant);
}

//==================fonction de mesure de la tension==================
Int mesure_tension()
{
      floattension,tensionPRCENT;
      int a;

        a = analogRead(A8);                  //stockage de la valeur récupéré dans la variable « charge »
      tension = a*(3.3/1024);               //conversion de la valeur lue (binaire) en volt
      if(tension>2.64)                      // si la tension est entre 4 et 5V cad ( entre 0 et 100%)
 {
        tensionPRCENT=(tension*100)/3.3;        // mettre sous forme de pourcentage avec 0v=0%
        val_tension=(tensionPRCENT-80)*5;       //adaptation du cahier de charge 4V=0% et avoir un
                                                //pourcentage en forme INTEGER
 }
else                          //si c'est inferieur 4V on affiche 0
 {
      tensionPRCENT=0;
      val_tension=0;
 }
return (val_tension);

}

Void mesure_temperature()
{
floatvolt,temp,vi;

  vi=analogRead(A10);              //stockage de la valeur récupérée dans la variable « vi »
  volt=(vi*5)/1024;           //règle de trois pour avoir la valeur correspondante a la tension
if(volt>=1.88 && volt<2.74)    //si la tension est comprise dans la plage qui correspond aux valeurs de
                                 //températures voulu.
 {
          temp=(volt-1.88);
          temp=(temp*10000)/86;   //température en fonction du nombre de pas
          val_temperature=temp*1;          //avoir la valeur en forme « entier »
 }
}

/**********************fonction de définition de l'interruption*************************/
voidstartTimer(Tc *tc, uint32_t channel, IRQn_Typeirq, uint32_t frequency) {
pmc_set_writeprotect(false);
pmc_enable_periph_clk((uint32_t)irq);
TC_Configure(tc, channel, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK4);
uint32_trc = VARIANT_MCK/128/frequency; //128 because we selected TIMER_CLOCK4 above
TC_SetRA(tc, channel, rc/2); //50% high, 50% low
TC_SetRC(tc, channel, rc);
TC_Start(tc, channel);
tc->TC_CHANNEL[channel].TC_IER=TC_IER_CPCS;
tc->TC_CHANNEL[channel].TC_IDR=~TC_IER_CPCS;
NVIC_Setpriority(irq, 1);
NVIC_EnableIRQ(irq);
}
```

CONCLUSION

Au terme de notre grand projet, nous pouvons affirmer que, celui-ci à été très bénéfique pour nous dans la mesure où il nous a permis d'acquérir de nouvelles connaissances et par ailleurs, d'asseoir quelques acquis que nous avons reçus tout au long de notre formation.

En outre, ce Grand Projet nous a aussi permis de développer la rigueur dans le travail, un esprit critique sur les différentes étapes de notre projet en vue d'apporter toujours la solution la mieux appropriée mais aussi un sens du dynamisme de travail en équipe car rappelons-le, ce projet a réunit plusieurs corps des métiers au sein de l'IUT.

En sommes, nous pouvons dire que le cahier de charges qui nous a été confié, pour ce projet, n'a pas été réalisé dans sa totalité, néanmoins nous avons pu réaliser une partie importante de celui-ci. Nous estimons qu'il y'a toujours des améliorations qui peuvent être faites sur ce travail.

LISTE DES FIGURES

LISTE DES ANNEXES

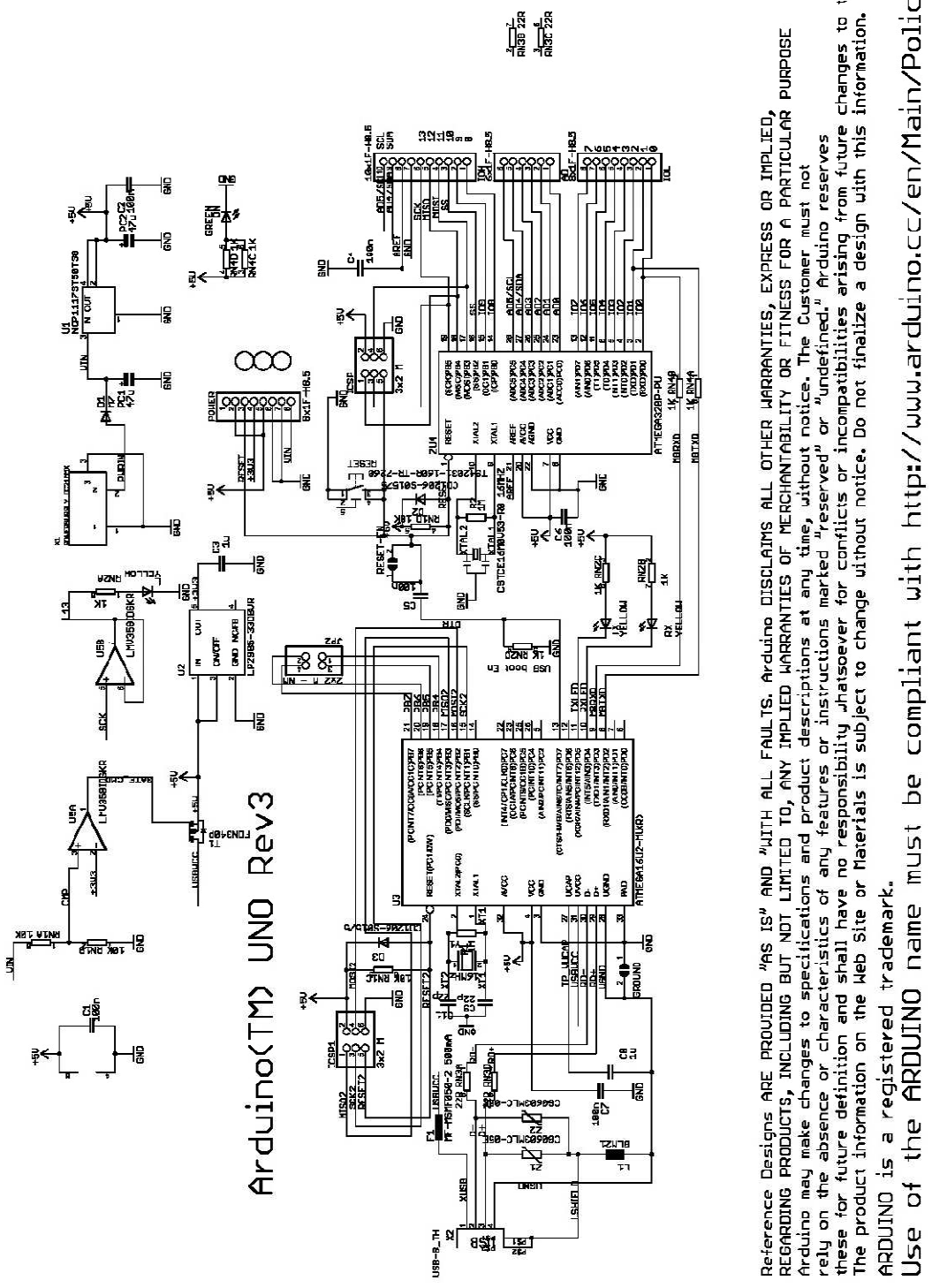
REFERENCES BIBLIOGRAPHIQUES ET WEBOGRAPHIQUES

- Julien Bayle, PACKT Publishing, C programming for arduino
- Ph. D. Jack Purdum, TECHNOLOGY IN ACTION, Beginning C for Arduino
- Alan Trevennor, TECHNOLOGY IN ACTION, Pratical AVR Microcontrollers
- Michael Margolis, O'REILLY, Arduino CookBook 2nd Edition

- Arduino Due references, http://arduino.cc/en/Main/ArduinoBoardDue#.Uyq2Zqh5OSo
- Arduino Uno references, http://arduino.cc/en/Main/ArduinoBoardUno
- SDfat pour arduino Due, http://forum.arduino.cc/index.php?PHPSESSID=haq7p1secc27ra6n1o2kb5mnc0&topic=135439.15
- Programmation arduino, http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ARDUINO
- Tutoriel Xbee, http://jeromeabel.net/fr/ressources/xbee-arduino
- Microshield SD Arduino, https://www.sparkfun.com/products/9802

ANNEXES

ANNEXE 1 : Schéma structurel de la carte Arduino UNO



ANNEXE 2 : Les fonctions NVIC (NestedVectorInterrupt Controller) utilisé pour la programmation des interruptions sur le SAM3X

12.20.10.1 NVIC programming hints

Software uses the CPSIE I and CPSID I instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

Table 12-29. CMSIS functions for NVIC control

CMSIS interrupt control function	Description
void NVIC_SetPriorityGrouping(uint32_t priority_grouping)	Set the priority grouping
void NVIC_EnableIRQ(IRQn_t IRQn)	Enable IRQn
void NVIC_DisableIRQ(IRQn_t IRQn)	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)	Return true if IRQn is pending
void NVIC_SetPendingIRQ (IRQn_t IRQn)	Set IRQn pending
void NVIC_ClearPendingIRQ (IRQn_t IRQn)	Clear IRQn pending status
uint32_t NVIC_GetActive (IRQn_t IRQn)	Return the IRQ number of the active interrupt
void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)	Set priority for IRQn
uint32_t NVIC_GetPriority (IRQn_t IRQn)	Read priority of IRQn
void NVIC_SystemReset (void)	Reset the system

For more information about these functions see the CMSIS documentation.

ANNEXE 3 : Le TimerCounter du microcontrôleur SAM3X

37. Timer Counter (TC)

37.1 Description

The Timer Counter (TC) includes three identical 32-bit Timer Counter channels.

Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

The Timer Counter (TC) embeds a quadrature decoder logic connected in front of the 3 timers and driven by TIOA0, TIOB0 and TIOA1 inputs. When enabled, the quadrature decoder performs the input lines filtering, decoding of quadrature signals and connects to the 3 timers/counters in order to read the position and speed of the motor through user interface.

The Timer Counter block has two global registers which act upon all three TC channels.

The Block Control Register allows the three channels to be started simultaneously with the same instruction.

The Block Mode Register defines the external clock inputs for each channel, allowing them to be chained.

Table 37-1 gives the assignment of the device Timer Counter clock inputs common to Timer Counter 0 to 2.

Table 37-1. Timer Counter Clock Assignment

Name	Definition
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5[1]	SLCK

Note: 1. When Slow Clock is selected for Master Clock (CSS = 0 in PMC Master Clock Register), TIMER_CLOCK5 input is equivalent to Master Clock

37.2 Embedded Characteristics

- Three 32-bit Timer Counter Channels
- A Wide Range of Functions Including:
  - Frequency Measurement
  - Event Counting
  - Interval Measurement
  - Pulse Generation
  - Delay Timing
  - Pulse Width Modulation
  - Up/down Capabilities

Table 37-2. Signal Name Description

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output
	TIOB	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Input/Output
	INT	Interrupt Signal Output
	SYNC	Synchronization Input Signal

37.4 Pin Name List

Table 37-3. TC Pin List

Pin Name	Description	Type
TCLK0-TCLK2	External Clock Input	Input
TIOA0-TIOA2	I/O Line A	I/O
TIOB0-TIOB2	I/O Line B	I/O
FAULT	Drives internal fault input of PWM	Output

37.5 Product Dependencies

37.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the TC pins to their peripheral functions.

Table 37-4. I/O Lines

Instance	Signal	I/O Line	Peripheral
TC0	TCLK0	PB26	B
TC0	TCLK1	PA4	A
TC0	TCLK2	PA7	A
TC0	TIOA0	PB25	B
TC0	TIOA1	PA2	A
TC0	TIOA2	PA5	A
TC0	TIOB0	PB27	B
TC0	TIOB1	PA3	A
TC0	TIOB2	PA6	A
TC1	TCLK3	PA22	B
TC1	TCLK4	PA23	B
TC1	TCLK5	PB16	A

Table 37-4. I/O Lines (Continued)

TC1	TIOA3	PE9	A
TC1	TIOA4	PE11	A
TC1	TIOA5	PE13	A
TC1	TIOB3	PE10	A
TC1	TIOB4	PE12	A
TC1	TIOB5	PE14	A
TC2	TCLK6	PC27	B
TC2	TCLK7	PC30	B
TC2	TCLK8	PD9	B
TC2	TIOA6	PC25	B
TC2	TIOA7	PC28	B
TC2	TIOA8	PD7	B
TC2	TIOB6	PC26	B
TC2	TIOB7	PC29	B
TC2	TIOB8	PD8	B

37.5.2 Power Management

The TC is clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the Timer Counter clock.

37.5.3 Interrupt

The TC has an interrupt line connected to the Interrupt Controller (IC). Handling the TC interrupt requires programming the IC before configuring the TC.

37.5.4 Fault Output

The TC has the FAULT output connected to the fault input of PWM. Refer to Section 37.6.17 "Fault Mode", and to the product Pulse Width Modulation (PWM) implementation.

ANNEXE 4 : Configuration de la priorité sur le microcontroleur SAM3X

12.6.5 Exception priorities

As Table 12-9 on page 81 shows, all exceptions have an associated priority, with:

- a lower priority value indicating a higher priority
- configurable priorities for all exceptions except Reset, Hard fault.

If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities see

- "System Handler Priority Registers" on page 180
- "Interrupt Priority Registers" on page 164.

Configurable priority values are in the range 0-15. This means that the Reset, Hard fault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

12.6.6 Interrupt priority grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This divides each interrupt priority register entry into two fields:

- an upper field that defines the *group priority*
- a lower field that defines a *subpriority* within the group.

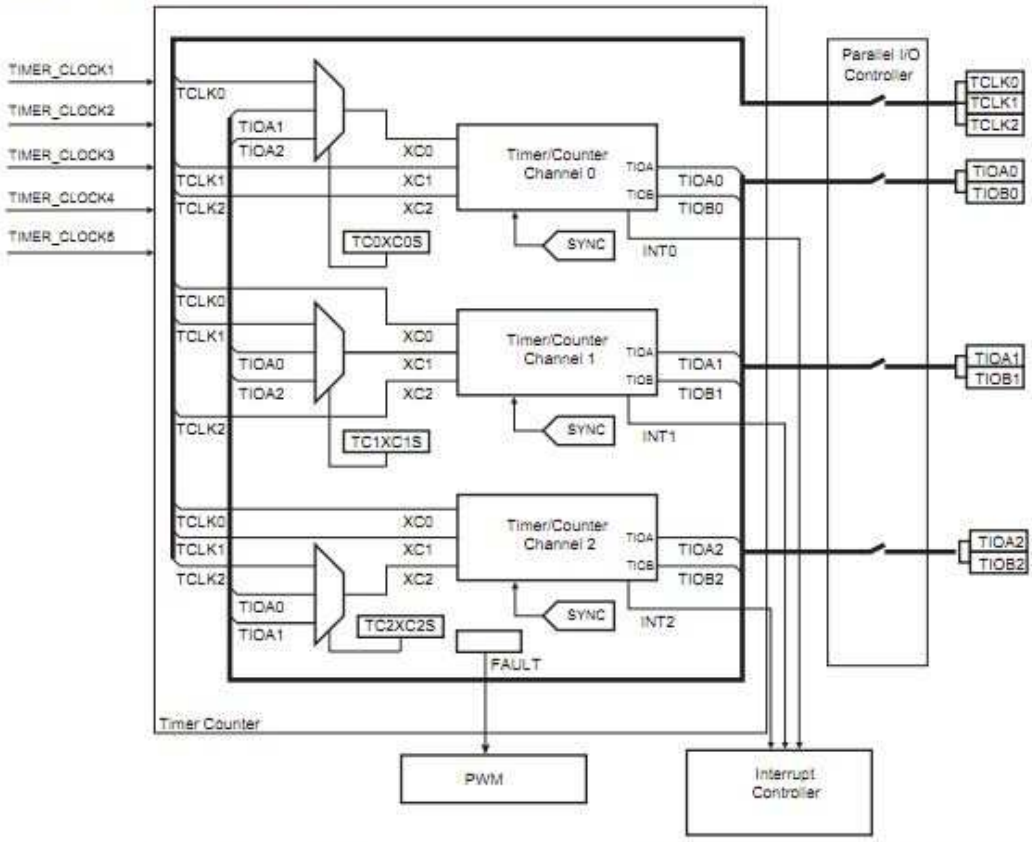
Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler,

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see "Application Interrupt and Reset Control Register" on page 176.

37.3 Block Diagram

Figure 37-1. Timer Counter Block Diagram



Note: The quadrature decoder logic connections are detailed in Figure 37-15 "Predefined Connection of the Quadrature Decoder with Timer Counters"

ANNEXE 5 : Caractéristiques du module XBEE Pro

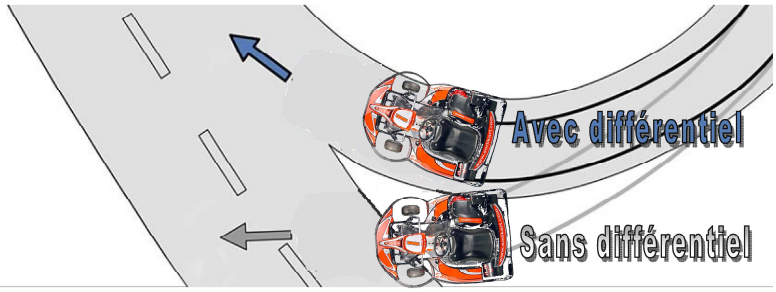
Platform	XBee	XBee-PRO
Performance		
Power output	1mW (+0 dBm) North American & International version	63 mW (+18 dBm) North American version 10 mW (+10 dBm) International version
Indoor/Urban range	Up to 100 ft (30 m)	Up to 300 ft (90 m)
Outdoor/RF line-of-sight range	Up to 300 ft (90 m)	Up to 1 mile (1.6 km) RF LOS
Receiver sensitivity	-92 dBm	-100 dBm (all variants)
RF data rate	250 Kbps	250 Kbps
Operating frequency	2.4 GHz	2.4 GHz
Interface data rate	Up to 115.2 Kbps	Up to 115.2 Kbps
Networking		
Spread spectrum type	DSSS (Direct Sequence Spread Spectrum)	
Supported network topologies	Point-to-point, point-to-multipoint, & peer-to-peer	
Error handling	Retries & acknowledgements	
Filtration options	PAN ID, Channel, and 64-bit addresses	
Channel capacity	16 Channels	12 Channels
Addressing	65,000 network addresses available for each channel	
Power		
Supply voltage	2.8 - 3.4 VDC XBee Footprint Recommendation: 3.0 - 3.4 VDC	2.8 - 3.4 VDC XBee Footprint Recommendation: 3.0 - 3.4 VDC
Transmit current	45 mA (@ 3.3 V) boost mode 35 mA (@ 3.3 V) normal mode	215 mA (@ 3.3 V)
Receive current	50 mA (@ 3.3 V)	55 mA (@ 3.3 V)
Power-down sleep current	<10 ÂµA at 25Â° C	<10 ÂµA at 25Â° C

ANNEXE 6 : Posters rélisés pour la promotion du Kart sans pilote du département GEII

KART sans pilote du département GEII

Le différentiel électronique

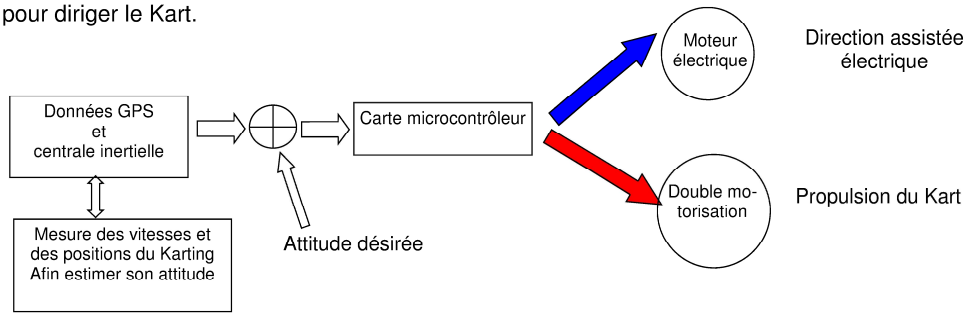
Le différentiel est un système utilisé dans l'automobile pour donner plus de tenue de route à un véhicule surtout dans les virages. Il a été utilisé depuis plusieurs années sur les monoplaces de Formule 1 car il élimine le patinage des roues en sortie de virage.



Dans le cas du Karting, ce différentiel emploie deux moteurs indépendants sur les roues arrières lui permettant d'être plus performant surtout en temps de pluie.

Le pilotage automatique

Avec les données de la centrale inertielle et du module GPS, la carte Raspberry Pi pilote séparément les moteurs électriques de propulsion ainsi que le moteur de la direction assistée électrique pour diriger le Kart.



Master GSAT

KART sans pilote du département GEII

- Vitesse max: 35km/h
- Budget du projet d'environ 1000 Euros
- Alimentation: 2 batteries 36V 14Ah et une batterie de 12V 20Ah pour la direction assistée

Centrale inertielle

- Dispose de 1 gyroscope 3 axes ,1 magnétomètre 3 axes et 1 accéléromètre 3 axes pour connaître l'altitude du Kart, sa vitesse ainsi que sa position sur le circuit

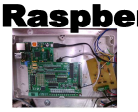
Module GPS

- Permet de situer le Kart sur le circuit avec une précision de 10m .Il est utilisé pour compenser la dérive de la position estimée par la centrale inertielle

Capteur d'angle du volant

Le capteur est équipé d'un potentiomètrepermettant de mesurer la position du volant

Carte Raspberry Pi



Nano-ordinateur disposant d'un processeur et d'une RAM, il constitue le cœur de tout le système. Il reçoit les données des capteurs, de la centrale inertielle ainsi que du module GPS et réagit en conséquence sur la direction assistée et sur les variateurs de chaque moteur

Capteurs de vitesses

Les deux capteurs sont fixés aux deux arbres de transmission pour fournir une mesure de vitesse de chaque roue à la Raspberry

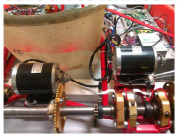
Variateurs

Permettant de contrôler les vitesses des 2 moteurs arrière droit et gauche

Direction

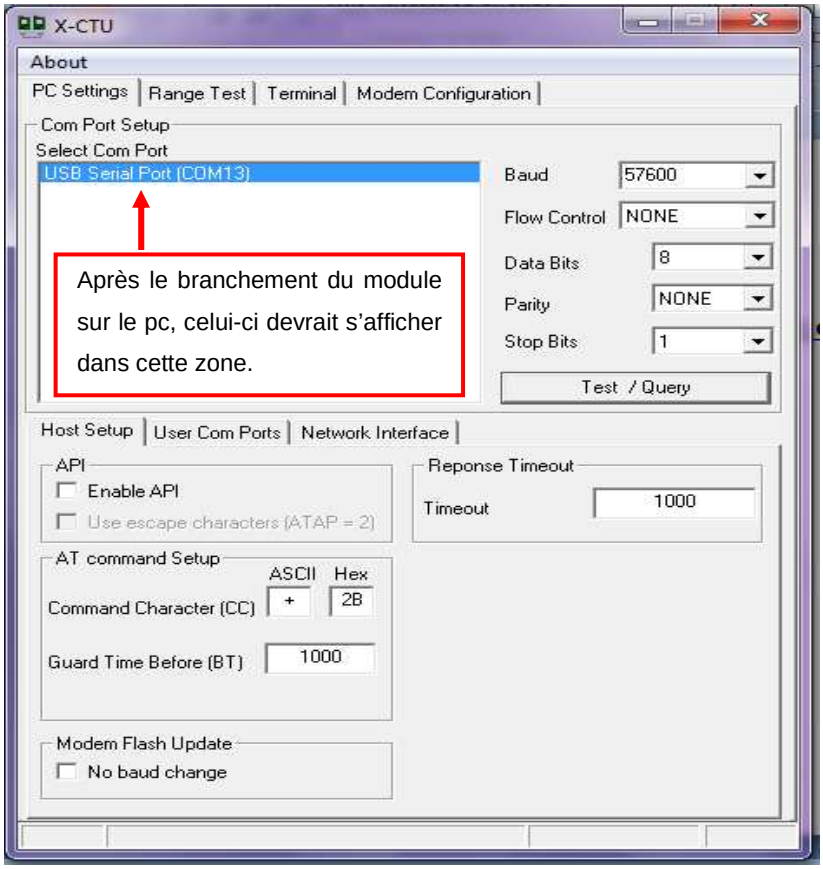
Système de direction assistée récupérée d'une Twingo. Commandée par la Raspberry à travers un dispositif de pont en H.

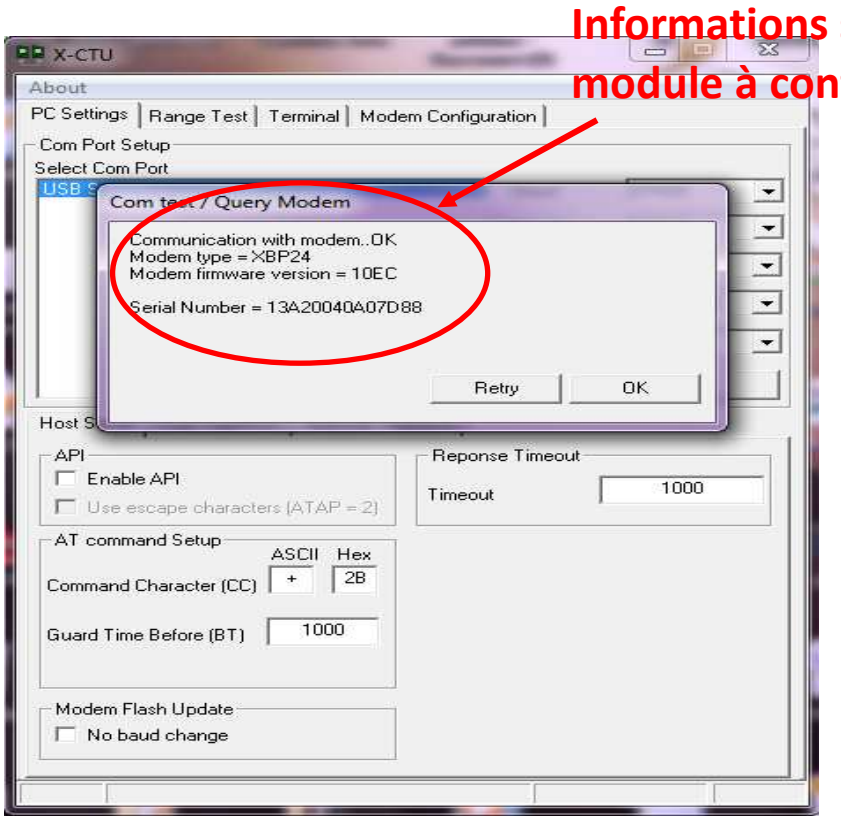
Moteurs à courant continu



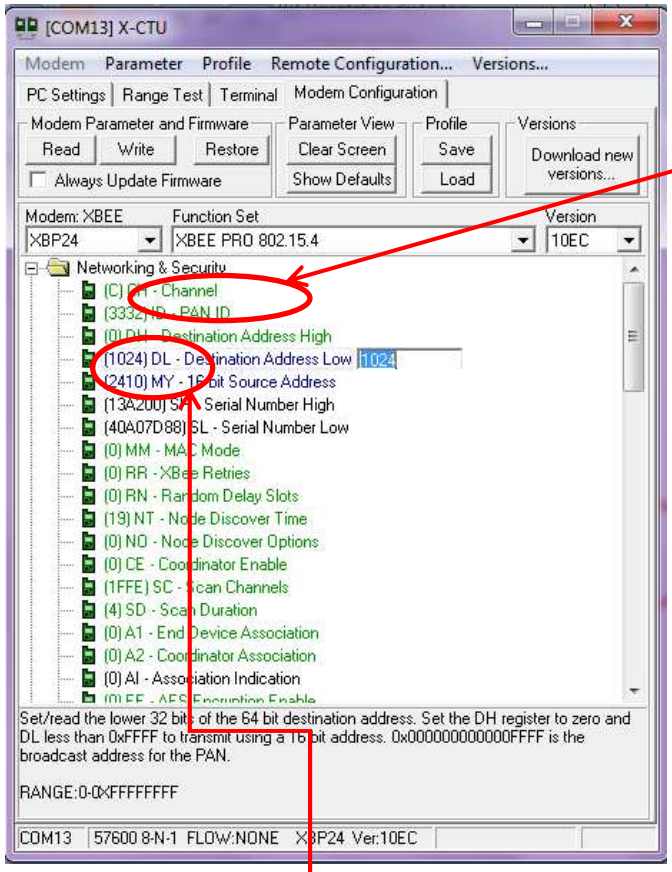
ANNEXE 7 : Configuration d'un module Xbee Pro en point à point sur le logiciel X-CTU

- Le logiciel X-CTU pour la configuration des modules Xbee Pro





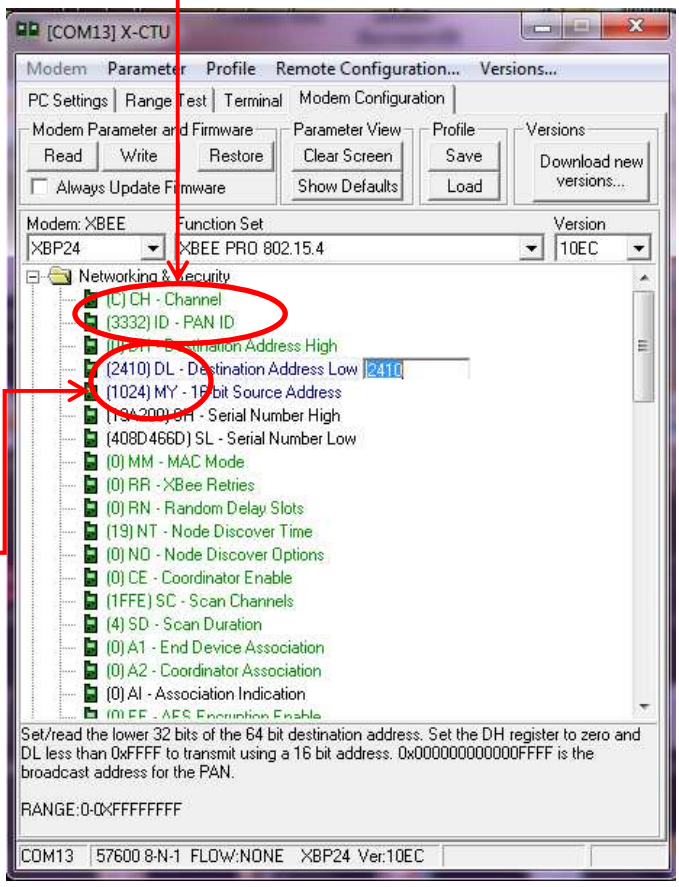
- Les éléments à configurer



Module émetteur

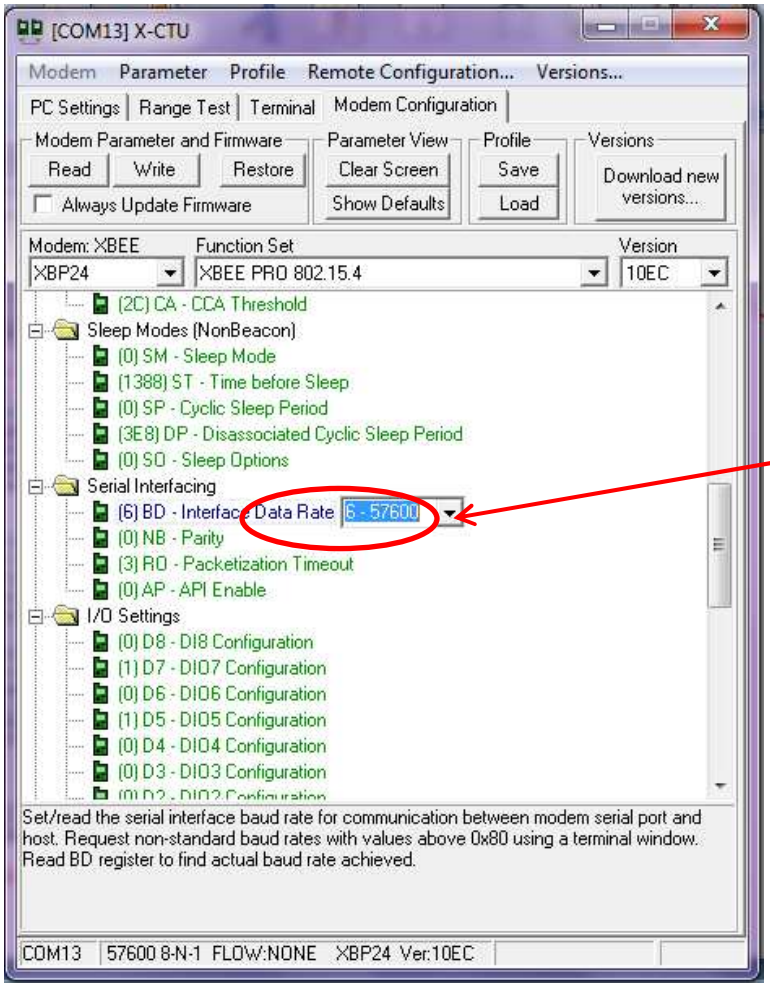
Le même PAN ID pour les 2 modules

DL émetteur = MY du récepteur
MY émetteur = DL du récepteur
Sur 2 octets en Hexadécimal



Module récepteur

Détermination de la vitesse de communication entre les deux modules



Configurer la même vitesse de communication sur les deux modules (57600 Bauds pour notre utilisation)