Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS

Institut Universitaire de Technologie

Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Expression technique

Récupération de l'état des batteries de plusieurs karting depuis un ordinateur

CALDEIRA Mathieu
MBETIBANGA kévin
2013 – 2014
K4

Enseignants :
Thierry LEQUEU
Philippe AUGER

Université François-Rabelais de Tours
Institut Universitaire de Technologie de Tours
Département Génie Électrique et Informatique Industrielle

UNIVERSITE FRANCOIS-RABELAIS
TOURS

Institut Universitaire de Technologie

Département
GENIE ELECTRIQUE ET
INFORMATIQUE INDUSTRIELLE

Expression technique

Récupération de l'état des batteries de plusieurs karting depuis un ordinateur

CALDEIRA Mathieu
MBETIBANGA kévin
2013 – 2014
K4

Enseignants :
Thierry LEQUEU
Philippe AUGER

Table des matières

Introduction

Monsieur Tartempion désire récupérer la tension des batteries de plusieurs kartings dans son hangar sur un ordinateur. Pour résoudre ce dilemme, il nous a donc proposé d'étudier son projet.

Pour l'aider, dans le cadre de l'étude et réalisation, nous avons donc décidé de nous même fabriquer des modules qui nous permettrons de récupérer l'état des batteries sur un ordinateur. Pour cela nous allons faire plusieurs cartes électroniques (maître et esclave) qui vont permettre d'établir la communication et faire un logiciel pour nous permettre de traiter les données fournies par ces deux éléments. Pour les réaliser, nous avons donc dû respecter certaines contraintes :

- Terminer le projet pour mi-novembre.
- Ne pas dépasser un budget de 50€
- Respecter l'encombrement pour les cartes électroniques
- Avoir une communication d'une longue portée (5 à 10 mètres)
- Essayer d'utiliser les mêmes composants pour réduire les coûts
- Éviter les perturbations extérieures lors de la conception des cartes

1. Planning prévisionnel

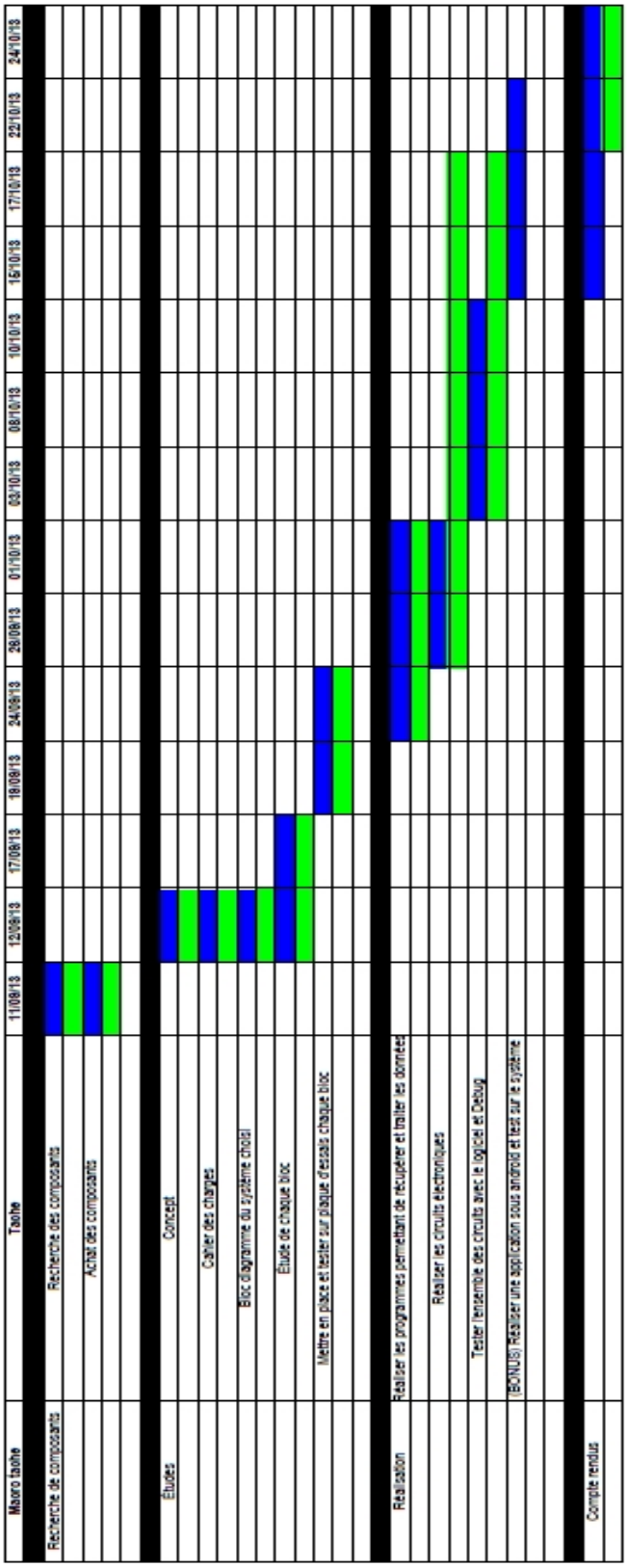


Illustration 1: Planning prévisionnel

2. Études

2.1. Recherche d'une solution pour avoir le fonctionnement désiré

Le but de notre projet étant de récupérer les informations depuis un ordinateur, il fallait donc pouvoir faire en sorte que la partie électronique puisse communiquer avec l'ordinateur, pour cela nous avions divers choix :

- Utiliser un micro-contrôleur embarquant ce qu'il faut pour gérer une liaison USB
- Utiliser un port série
- Utiliser un câble convertisseur USB > Série

Vu que nous ne savons pas si l'ordinateur de M. Tartempion possédait un port série ou pas et qu'il était tout de même plus accessible d'utiliser une liaison série que USB, nous avons donc décidé d'utiliser un câble USB-série qui permet de créer un port série virtuel. Ce qui permet également de simplifier la communication avec un micro-contrôleur il est facile de communiquer par liaison série.

Pour ce qui est de la communication entre la carte qui est reliée à l'ordinateur de l'opérateur et les cartes qui sont reliées sur les kartings nous avons décidé de mettre en place une liaison radio basée sur un système Maître-Esclave ou le maître interroge chaque esclave pour récupérer la tension des différentes batteries. Pour cela nous avons plusieurs choix au niveau des modules radios :

- Utiliser un module radio 433Mhz classique

Illustration 2: Module radio 433Mhz « classique »

Ces modules ont la particularité d'être simple à utiliser puisque l'on peut encore une fois leur envoyer des données par liaison série. En revanche, ces modules ne permettent pas de vérifier si le destinataire à bien reçu les données et il ne garantit pas que les données arrivent sans erreur.

- Utiliser un module radio 2.4 GHz à base de NRF24L01

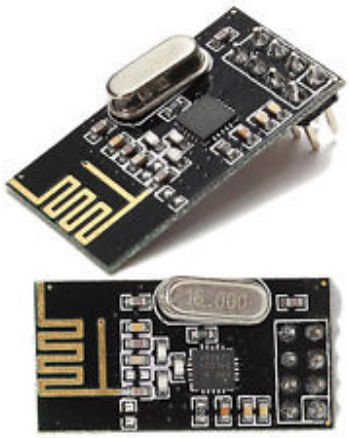


Illustration 3: Modules NRF24L01

Ces modules radio contrairement aux modules classiques garantissent le fait que le destinataire reçoit les données sans erreur et indiquent le fait que le récepteur a reçu les données. Ce module prend en charge également un tas de choses comme la possibilité d'envoyer des données à une certaine adresse. On peut changer la vitesse, le nombre de données qu'on envoie à chaque fois. En revanche ces modules sont plus difficiles à utiliser puisqu'on est obligé d'envoyer beaucoup de commandes par liaison SPI pour les paramétrer.

Notre choix s'est porté sur le deuxième module puisque dans notre cas nous aurons plusieurs cartes à utiliser. Ce qui permettra de faciliter le système d'adressage. De plus nous n'aurons pas à gérer nous même les contrôles d'erreur puisque si la donnée est envoyée nous sommes sur qu'elle arrivera intacte vue que le module vérifie lui-même les erreurs. Connaissant ce module pour l'avoir déjà utilisé la contrainte, nous n'avons pas de difficultés à mettre en œuvre ce module.

Illustration 4: www.atmel.com

Pour ce qui est du micro-contrôleur qui gère les différentes cartes, nous avons plusieurs familles (STmicro-electronics, Microchip, Atmel…). Nous avons décidé d'utiliser des micro-contrôleurs de la famille Atmel[1] qui sont très utilisés à l'IUT et dont nous avons déjà a disposition le programmateur. Il est ainsi plus simple pour nous d'utiliser cette famille de micro-contrôleur.

Pour réaliser la carte esclave, nous avons besoin de 4 entrées analogiques qui vont nous permettre de récupérer la tension des batteries. De plus nous avons également besoin de beaucoup d'entrées et sorties numériques notamment pour les boutons qui servent à régler l'adresse de la carte. Pour cela nous avons décidé d'utiliser l'Atmega8 qui possède ces entrées analogiques et numériques.

Pour concevoir la carte maître, on avait d'abord choisi d'utiliser un AtTiny, car sur cette carte on n'avait pas besoin d'entrées analogiques (ADC). De plus on n'avait pas besoin de beaucoup d'entrées et sorties numériques pour cette carte. Cependant on nous a proposé d'utiliser le même composant (Atmega8) pour les deux cartes afin de réduire les coûts (ça permet de ne commander que un composant en grande quantité).

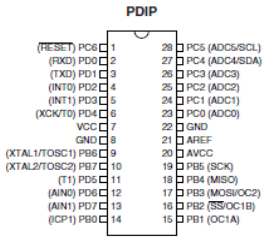


Illustration 5: ATmega8, www.atmel.com

2.2. Fonctionnement d'une liaison SPI

Nous allons vous expliquer le fonctionnement d'une liaison SPI (Serial Peripheral Interface). Une liaison SPI s'établit entre un maître et un esclave, ce qui permet d'avoir un échange de données par série entre ces derniers. L'échange de données peut s'effectuer dans les deux sens

(Full duplex). Cette liaison est synchrone, elle s'effectue par l'intermédiaire de trois fils notés MOSI (Master Output Slave Input), MISO (Master Input Slave Output) et SCK (SPI Serial Clock). La fréquence de l'horloge SCK est fixé par le maître et est programmable. La figure ci-dessous nous montre une liaison SPI classique entre le micro-contrôleur ″maître″ et plusieurs périphériques ″esclaves″. Dans ce cas, le maître peut sélectionner l'esclave avec qu'il veut communiquer.

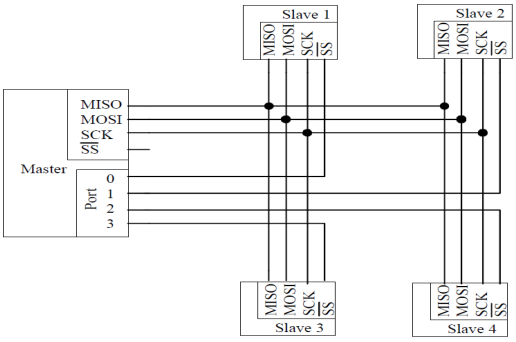


Illustration 6: Représentation d'une liaison SPI, http://bacstielectronique.free.fr

2.3. Fonctionnement d'une liaison série

La communication série nécessite trois fils au minimum: une masse pour référencer les signaux, un fil émetteur et un fil récepteur. Notre liaison série est en effet full-duplex, c'est à dire que l'on peut émettre et recevoir en même temps (comme le téléphone par exemple).

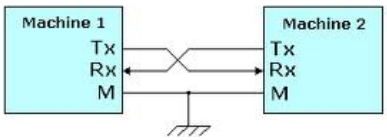


Illustration 7: Représentation d'une liaison série, http://sitelec.org

La différence principale entre le port parallèle et le port série est que les informations ne sont pas transmises simultanément sur des fils séparés mais les unes après les autres sur un même fil. Cela amène une économie de câble (un fil au lieu de 8). Ces bits sont utiles pour la synchronisation de l'émetteur et du récepteur.

En effet, la liaison série[2] est totalement asynchrone. Aucune horloge n'est transmise. Il faut donc se mettre d'accord sur la vitesse de transfert des bits et rajouter des bits de synchronisation.

Voici un petit résumé des différents paramètres rentrant en jeu lors d'une communication série :

- **longueur de mot:** sur le PC, le BIOS ne permet une longueur de mot que de 7 ou 8 bits.

- **Parité:** le mot transmis peut être suivi d'un bit de parité qui sert à détecter les erreurs éventuelles de transmission. Il existe deux parités : la parité paire et la parité impaire. Dans le cas de la parité paire, et pour le mot 10110101 contenant 5 états à 1, le bit de parité sera 1 amenant ainsi le nombre total de 1 à un nombre pair (6). Dans le cas de la parité impaire, le bit de parité aurait été 0, car le nombre total de 1 est déjà impair. L'intérêt de ce rajout est le suivant : si jamais lors de la transmission un état 1 est transformé en état 0 (perturbation du canal par des parasites par exemple) le nombre total de 1 change et donc le bit de parité recalculé par le récepteur ne correspond plus à celui reçu. L'erreur est donc détectée. Évidemment, si deux états à 1 passent à 0, l'erreur ne sera pas détectée mais la probabilité pour que cela arrive est très faible.

- **Bit de start:** lorsque rien ne circule sur la ligne, celle-ci est à l'état haut. Pour indiquer qu'un mot va être transmis, la ligne passe à bas avant de commencer le transfert. Cette précaution permet de resynchroniser le récepteur.

- **Bits de stop:** ces bits signalent la fin de la transmission. Selon le protocole utilisé, il peut y avoir 1, 1.5, ou 2 bits de stop (ces bits sont toujours à 1).

- **Vitesse de transmission:** la plupart des cartes série permettent de choisir une vitesse entre 300 et 9600 bauds (par exemple à 300 bauds, un bit est transmis tout les un trois-centième de seconde). Les cartes récentes proposent des vitesses jusqu'à 115200 bauds. Ces vitesses ne vous paraissent peut-être pas énormes mais il faut garder à l'esprit que la liaison série est avant tout pensée pour les liaisons téléphoniques par modems, dont la bande passante est très limitée.

2.4. Récupération de la tension des batteries

Pour récupérer une tension depuis un ATmega8 on doit lire une entrée ADC (entrée analogique), mais sa tension ne doit aller en principe que de 0V à 3.3V, par conséquent il fallait pouvoir adapter le niveau de tension qui arrive pour ne pas dépasser les 3.3V, pour cela nous avons utilisé un simple pont diviseur de tension[3].

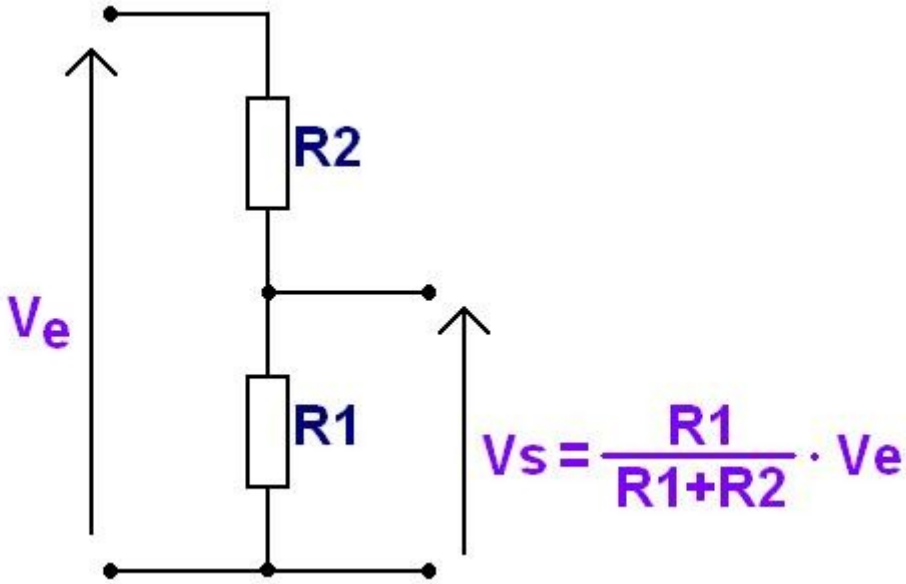


Illustration 8: Pont diviseur de tension, http://astuces-pratiques.fr

Pour les calculs nous avons fixé R2 à 10kOhms et nous avons définis la résistance R1 de façon à ne pas dépasser 3.3V en fonction de la tension que l'on prélève, pour être sur de ne pas dépasser, on fait comme si la tension d'une batterie était de 16V au lieu de 12V (car en réalité il y a plus de 12V)

Voici donc les valeurs de R1 que l'on a pris :

16V	32V	48V	64V
R1=2598 Ohms	R1=1149 Ohms	R1=738 Ohms	R1=543 Ohms

Nous avons également rajouté un condensateur sur les entrées analogiques pour éviter les parasites.

Nous devrions faire également un filtrage numérique mais ce n'est actuellement pas mis en place, le principe du filtrage numérique est de prendre plusieurs points de mesures, et d'en faire la moyenne.

2.5. Dialogue avec le module radio NRF24L01

Afin de pouvoir envoyer des données en passant par le module radio, il faut d'abord le configurer mais aussi s'adapter a son protocole de communication[1]. Comme nous l'avons vue précédemment, pour communiquer avec le module radio il faut utiliser une liaison SPI,

Des que le module est mis sous tension, il faut le configurer avant de pouvoir discuter avec, pour cela on doit configurer avec toute une série de registres[4].

1 Protocole : Un protocole est une façon de dialoguer entre deux parties, il faut que les deux parties discutent de la même façon pour se comprendre.

Nous allons d'abord commencer par expliquer le dialogue « physique » avec le module :

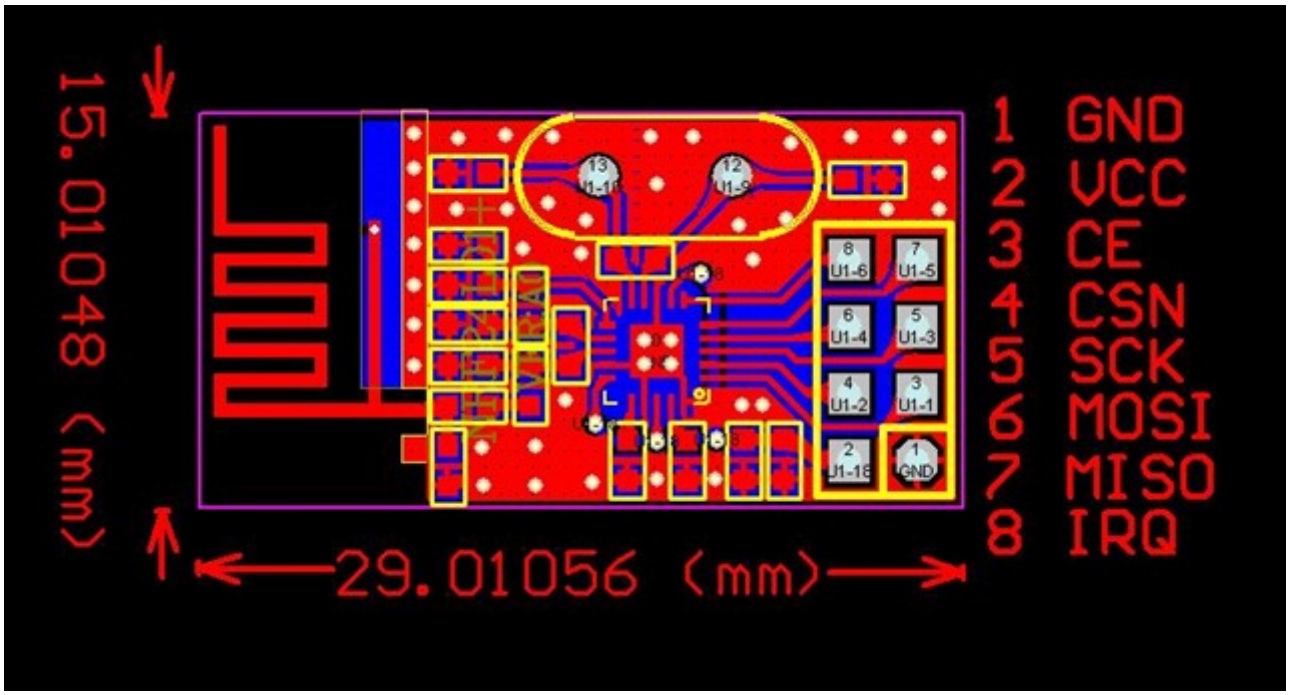


Illustration 9: Module NRF24L0, http://gizmosnack.blogspot.fr

- Les broches VCC et GND servent à la liaison SPI
- Les broches MISO, MOSI, SCK et CSN servent pour la liaison SPI, sur la broche MISO il y a les informations qui vont du module radio jusqu'au micro-ordinateur alors que sur la broche MOSI on a les informations qui vont du micro-contrôleur jusqu'au module radio, la broche SCK est celle où est envoyée le signal d'horloge créer par le micro-ordinateur pour cadencer la liaison SPI, quand t'a la broche CSN c'est celle qui sert a « sélectionner » le périphérique SPI avec lequel on souhaite discuter (puisque à titre de rappel, la liaison SPI permet de dialoguer avec plusieurs périphériques à tours de rôle), cette broche doit être mise a 0 pour communiquer avec le module radio.
- La broche CE, cette broche est mise à 1 uniquement quand on veut envoyer ou recevoir des données, (le module renvoie alors les données reçues qu'il a en mémoire (s'il en a) et le micro-contrôleur doit y transmettre les données à envoyer), quand cette broche est à 1 le module est dans un état « bloqué » il faut la repasser à 0 pour qu'il puisse se remettre à écouter et c'est au passage à 0 qu'il envoie ce que le micro-contrôleur lui avait envoyé.
- La broche IRQ, cette broche sert comme avertisseur configurable, elle peut permettre de nous avertir de l'arriver d'un message, de l'envoie d'un message ou d'une erreur de transmission (on peut configurer le module pour choisir l'information que l'on veut qu'elle nous donne) c'est une sortie en « tout ou rien ». Dans notre cas nous n'utilisons pas cette broche, car le micro-contrôleur demande lui-même au module s'il y a des données

reçues, ou l'état des transmissions, ça permet de réduire le câblage et de libérer une entrée sur le micro-contrôleur.

Ensuite concernant la configuration du module il faut regarder de près les différents registres, dans notre cas nous allons présenter uniquement les plus importants :

- Le registre « R_RX_PAYLOAD » : C'est celui auquel on fait appel pour récupérer les données que le module radio à reçu, (de son côté il enlève d'ailleurs ce qui a été récupérer) (de 1 a 32 octets (à définir dans un autre registre)) (LSB first)
- Le registre « W_TX_PAYLOAD » : C'est celui dans lequel on vient écrire les données qui seront à envoyer par liaison radio (de 1 à 32 octets (à définir dans un autre registre)) (LSB first)
- Le registre « CONFIG » : il sert notamment à régler les interruptions que l'on veut définir sur la sortie « IRQ » du module radio (pour qu'il avertisse par exemple de l'arrivée d'une trame, ou d'une erreur de transmission ou d'un envoie effectué) dans notre cas nous n'utilisons pas cette fonction, cependant on se sert de ce registre pour activer le module radio et d'activer la détection d'erreurs.
- Le registre « SETUP_AW » : il sert à choisir la taille de l'adresse des modules (de 3 octets à 5 octets), dans notre cas on utilise une taille de 5 octets mais il n'y a que 1 octet qui change.
- Le registre « SETUP_RETR » : permet d'activer ou non le fait que le module retransmette un message s'il n'a pas été reçu, avec le nombre de tentatives.
- Le registre « RF_CH » : permet de définir le canal sur lequel notre module fonctionne (il faut que les modules soient sur le même canal pour pouvoir communiquer entre eux)
- Le registre « RF_SETUP » : Il permet de définir la vitesse de transmission et la puissance d'émission.
- Le registre « STATUS » : C'est ici que l'on vient voir si l'on a reçu quelque chose, ou si la transmission a bien été effectuer, ou s'il y a eu une erreur, (dans le cas ou l'on n'utilise pas la broche « IRQ »)
- Le registre « RX_ADDR_P0 » : Permet de configurer l'adresse de réception n°1 du module (il peut y en avoir jusqu'à 6)
- Le registre « TX_ADDR » : Permet de configurer l'adresse d'émission du module (il n'y en a qu'une possible, mais peut être changée a tout moment)
- Le registre « RX_PW_P0 » : Permet de régler le nombre d'octets à envoyer/recevoir sur l'adresse de réception n°1

Vous pourrez retrouver en annexe la liste complète des registres de configuration.

Le module est donc soit récepteur, soit émetteur, donc dans notre cas ce que l'on fait c'est que pour les deux cartes (maître/esclave) le module est en permanence récepteur, mais il passe émetteur au moment de transmettre quelque chose, puis repasse récepteur. Nous n'utilisons qu'une

adresse de réception, quant à l'adresse d'émission dans le cas du maître, elle change en fonction du module que l'on interroge.

3. Réalisation

3.1. Réalisation de la carte « maître »

Le typon ci-dessous a été conçu sur Kicad avec 6 straps puis il était imprimé sur une plaque de cuivre avec des dimensions tout à fait acceptable, les pistes qui ont été bien configuré (0,8 mm). On a fait un plan de masse pour éviter tous types de parasites. Nous remarquons le composant principal (Atmega8) sera programmer pour nous permettre d'avoir une bonne communication avec la carte esclave. Suite à cela nous l'avons instauré dans un boîtier conforme à la taille de la carte.

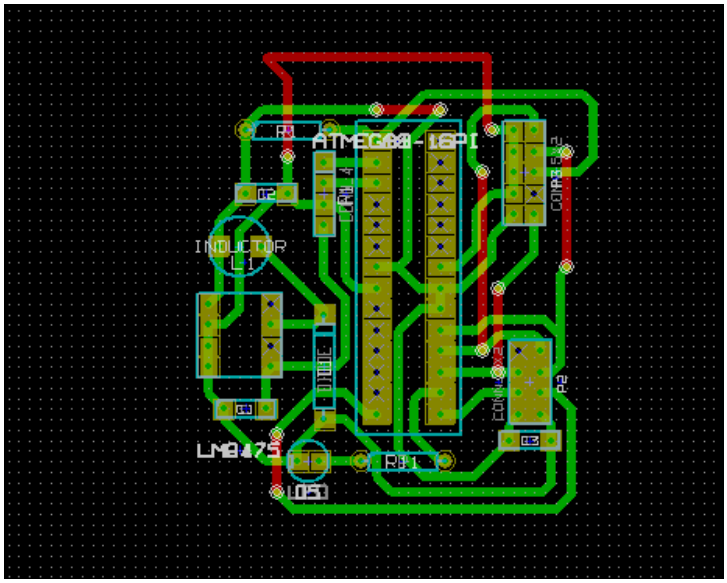


Illustration 10: Carte « maître »

Illustration 11: Carte « maître » dans son boîtier

3.2. Réalisation de la carte « esclave »

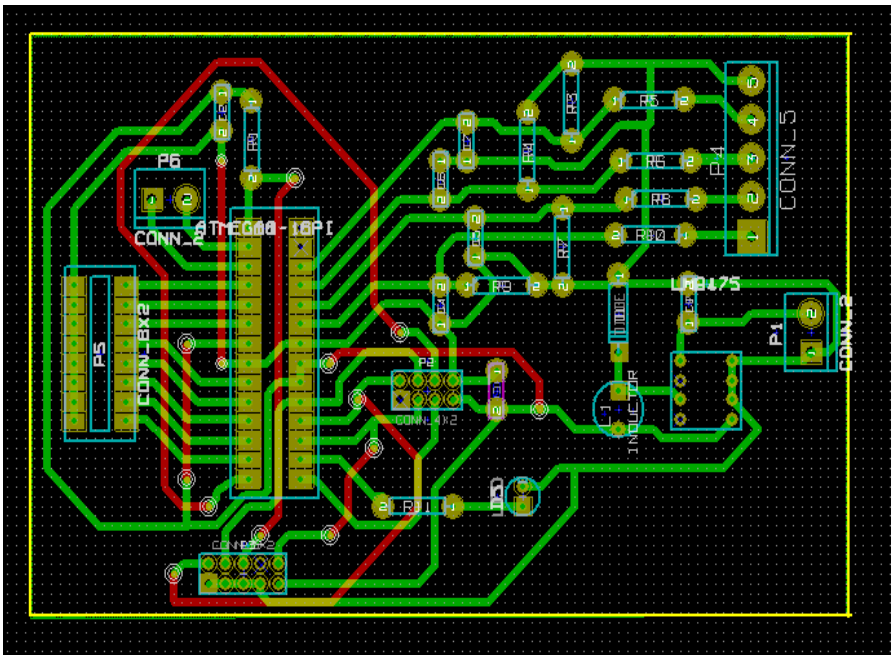


Illustration 12: Carte « esclave »

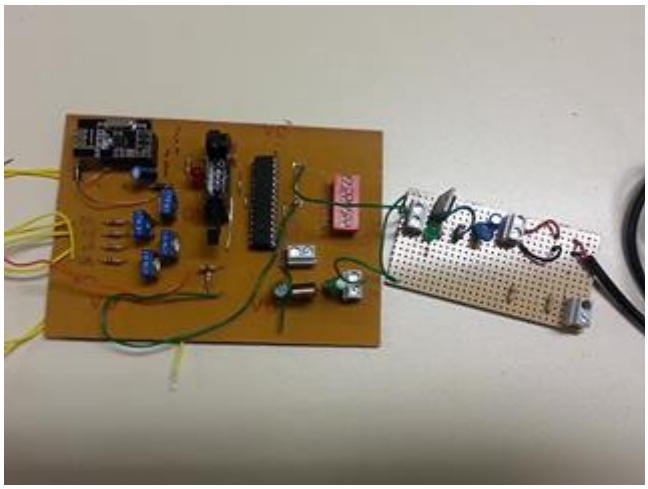


Illustration 13: Carte « esclave »

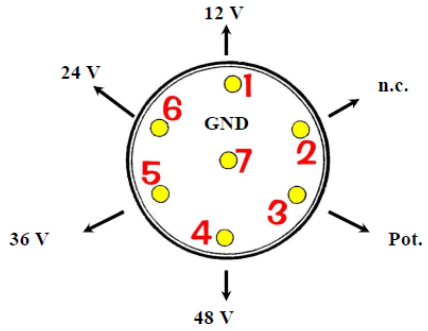
Cette carte aussi conçue sur Kicad. Nous avons aussi fait un plan de masse pour éviter les parasites. Cependant nous avons beaucoup de straps. Nous avons eu un problème de comptabilité électromagnétique. Il a été résolu par un condensateur qui n'arrivait pas découpler la tension. Nous avons donc pris un condensateur de découplage plus puissant pour éviter ce problème.

Pour avoir les tensions des batteries, le professeur nous a donné un connecteur qui est lié au port des batteries. Selon la datasheet du connecteur on a pu relier les bornes du connecteur[5] avec les entrées analogiques destinées pour chaque tension.

Illustration 14: Carte « esclave » dans son boîtier

Illustration 15: Bornier pour se raccorder au kart

3.1 Prise femelle sur le kart



1	6	5	4	3	2	7
+12 V	+24 V	+36 V	+48V	Curseur potentiomètre accélérateur	Non connecté	GND masse

Illustration 16: Prise femelle coté karting

3.3. Test des différents circuits

La carte maître fonctionne correctement. Elle a été programmée et testée. On a pu se rendre compte que cette carte arrivait à communiquer l'esclave à partir de son adresse. On a récupéré les données envoyer par l'esclave. La donnée qui était récupérée est un caractère envoyé par l'esclave.

La carte esclave fonctionne aussi correctement. Elle récupère la tension de chaque batterie. Cependant ces tensions ne sont pas assez précises, car la programmation est faite sous 8 bits. Il faut donc de filtrer numériquement les données et de mettre sous 10 bits pour être plus précis.

3.4. Réalisation du logiciel de contrôle

La réalisation du logiciel à été faite avec Builder C++, le logiciel permet de récupérer la tension des 4 batteries de chaque karting, donc son fonctionnement est simple : Il envoie par liaison série l'adresse du karting qu'il veut interroger (grâce à un port série virtuel qui est créé avec le cordon USB->Série), s'il reçoit 0 c'est que la transmission ne s'est pas faite, (il retente alors 3 fois), s'il reçoit 1 ça signifie que la transmission s'est faite, et donc il reçoit juste après, la tension de chaque batterie du karting en question.

La tension reçue est sous forme d'un nombre allant de 0 a 255 (8Bits) il suffit donc de faire un produit en croix pour récupérer la tension réel en Volts

Le programme permet également de rechercher lui-même les kartings plutôt que ce soit l'utilisateur qui les ajoute quant aux tensions de batteries qui sont récupérées, elles sont transmises à une base de données MySql que l'utilisateur peut ensuite exploiter pour en sortir un graphique.

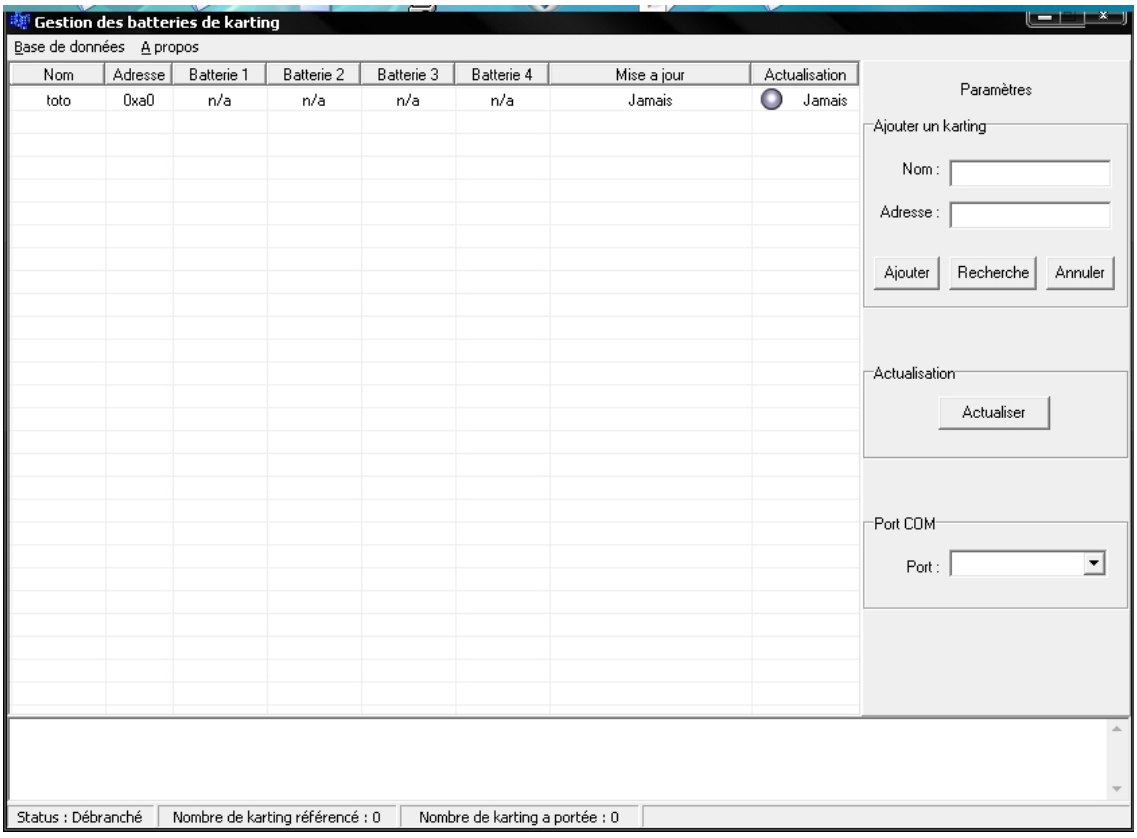


Illustration 17: Fenetre principale de l'application

Ici nous avons la fenêtre principale du programme, sur la droite nous avons ce qu'il faut pour ajouter un karting au programme, on peut donc ajouter manuellement un karting (en connaissant son adresse et en lui donnant un nom) ou faire une recherche automatique, le bouton actualiser sert à lancer la récupération des tensions quant à la partie « port COM » elle sert à choisir le port série que l'on souhaite utiliser pour communiquer.

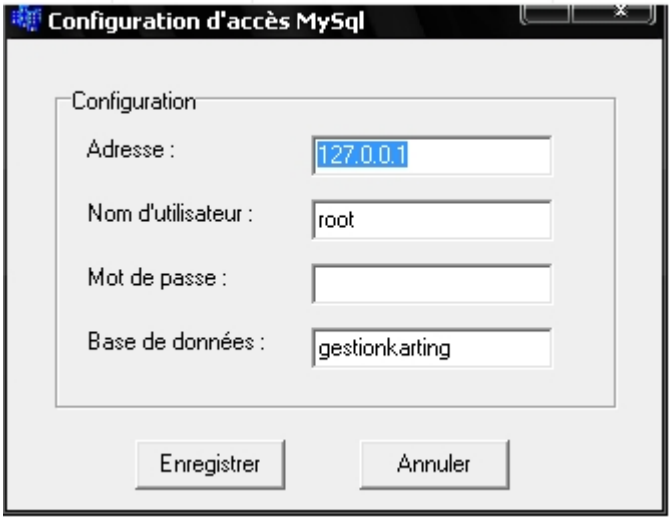


Illustration 18: Fenêtre de configuration MySql

Conclusion

Lors des séances de l'étude et réalisation, et en menant notre projet, nous avons pu mettre en œuvre des informations et des connaissances reçues lors des cours d'électroniques et d'informatique ainsi on a pu traduire des études théoriques en études pratiques. On a choisi ce sujet avec des composants étudiés auparavant. Cela nous a permis de se familiariser avec ce type de composants. Ce fût une expérience enrichissante, car nous avons pu réaliser un projet personnel de manière autonome, c'est-à-dire qu'à partir d'un budget nous devons réaliser un projet qui sera présenter. Ce projet nous a permis de rassembler des domaines différents. Nous avons beaucoup appris sur la récupération des données et le traitement de ces données venant du karting.

Résumé

Dans le cadre du cours d'études et réalisations, nous avons eu pour mission de récupérer la tension de chaque batterie du karting

Puisque notre projet est constitué de deux parties fondamentales, la réalisation de notre projet s'est déroulée en deux temps :

En premier temps nous avons réalisé la communication entre une carte maître et une carte esclave, dans cette partie notre but était de concevoir deux cartes électroniques qui nous permettent d'établir une communication. Ces cartes qui possèdent un micro-contrôleur (Atmega8) est programmé pour établir cette conversation entre ces cartes. À partir de cette programmation, on a pu voir qu'il y avait une bonne communication entre ces cartes.

Dans un deuxième temps nous avons fait un programme qui va nous permettre de récupérer la tension de chaque batterie du karting. Cette programmation est faite sur Builder en langage C++. Cette application permettra à l'opérateur de voir le karting qui a besoin d'être rechargé. À partie de cette application, il pourra récupérer la tension de chaque batterie de chaque karting. Puis toutes les informations récupérées sont dirigées dans une base de données MySql qui permettra de traiter ces données sous un graphique.

Annexe 1 : Registres de configuration du module NRF24L01

Command name	Command word (binary)	# Data bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read command and status registers. AAAAA = 5 bit Register Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write command and status registers. AAAAA = 5 bit Register Map Address Executable in power down or standby modes only.
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, that is, acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last transmitted payload. Packets are repeatedly retransmitted as long as CE is high. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deacti-vated during package transmission
ACTIVATE	0101 0000	1	This write command followed by data 0x73 acti-vates the following features: • R_RX_PL_WID • W_ACK_PAYLOAD • W_TX_PAYLOAD_NOACK A new ACTIVATE command with the same data deactivates them again. *This is executable in power down or stand by modes only.* The R_RX_PL_WID, W_ACK_PAYLOAD, and W_TX_PAYLOAD_NOACK features registers are initially in a deactivated state; a write has no effect, a read only results in zeros on MISO. To activate these registers, use the ACTIVATE com-mand followed by data 0x73. Then they can be accessed as any other register in nRF24L01. Use the same command and data to deactivate the registers again.
R_RX_PL_WID*	0110 0000		Read RX-payload width for the top R_RX_PAYLOAD in the RX FIFO.
W_ACK_PAYLOAD*	1010 1PPP	1 to 32 LSByte first	Used in RX mode. Write Payload to be transmitted together with ACK packet on PIPE PPP. (PPP valid in the range from 000 to 101). Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled using first in - first out principle. Write payload: 1– 32 bytes. A write operation always starts at byte 0.

Illustration 19: Registres de configuration du module radio, (datasheet)

Command name	Command word (binary)	# Data bytes	Operation
W_TX_PAYLOAD_NOACK*	1011 000	1 to 32 LSByte first	Used in TX mode. Disables AUTOACK on this specific packet.
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

Illustration 20: Registres de configuration du module radio, (datasheet)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the IRQ pin 0: Reflect RX_DR as active low interrupt on the IRQ pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the IRQ pin 0: Reflect TX_DS as active low interrupt on the IRQ pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the IRQ pin 0: Reflect MAX_RT as active low interrupt on the IRQ pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' – 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0:POWER DOWN
	PRIM_RX	0	0	R/W	RX/TX control 1: PRX, 0: PTX
01	EN_AA Enhanced ShockBurst™				Enable 'Auto Acknowledgment' Function Disable this functionality to be compatible with nRF2401, see page 65
	Reserved	7:6	00	R/W	Only '00' allowed
	ENAA_P5	5	1	R/W	Enable auto acknowledgement data pipe 5
	ENAA_P4	4	1	R/W	Enable auto acknowledgement data pipe 4
	ENAA_P3	3	1	R/W	Enable auto acknowledgement data pipe 3
	ENAA_P2	2	1	R/W	Enable auto acknowledgement data pipe 2
	ENAA_P1	1	1	R/W	Enable auto acknowledgement data pipe 1
	ENAA_P0	0	1	R/W	Enable auto acknowledgement data pipe 0
02	EN_RXADDR				Enabled RX Addresses
	Reserved	7:6	00	R/W	Only '00' allowed
	ERX_P5	5	0	R/W	Enable data pipe 5.
	ERX_P4	4	0	R/W	Enable data pipe 4.

Illustration 21: Registres de configuration du module radio, (datasheet)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	ERX_P3	3	0	R/W	Enable data pipe 3.
	ERX_P2	2	0	R/W	Enable data pipe 2.
	ERX_P1	1	1	R/W	Enable data pipe 1.
	ERX_P0	0	1	R/W	Enable data pipe 0.
03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' – 5 bytes LSByte is used if address width is below 5 bytes
04	SETUP_RETR				Setup of Automatic Retransmission
	ARD	7:4	0000	R/W	Auto Retransmit Delay '0000' – Wait 250µS '0001' – Wait 500µS '0010' – Wait 750µS …….. '1111' – Wait 4000µS (Delay defined from end of transmission to start of next transmission)[a]
	ARC	3:0	0011	R/W	Auto Retransmit Count '0000' –Re-Transmit disabled '0001' – Up to 1 Re-Transmit on fail of AA …… '1111' – Up to 15 Re-Transmit on fail of AA
05	RF_CH				RF Channel
	Reserved	7	0	R/W	Only '0' allowed
	RF_CH	6:0	0000010	R/W	Sets the frequency channel nRF24L01 operates on
06	RF_SETUP				RF Setup Register
	Reserved	7:5	000	R/W	Only '000' allowed
	PLL_LOCK	4	0	R/W	Force PLL lock signal. Only used in test
	RF_DR	3	1	R/W	Air Data Rate '0' – 1Mbps '1' – 2Mbps
	RF_PWR	2:1	11	R/W	Set RF output power in TX mode '00' – -18dBm '01' – -12dBm '10' – -6dBm '11' – 0dBm
	LNA_HCURR	0	1	R/W	Setup LNA gain

Illustration 22: Registres de configuration du module radio, (datasheet)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
07	STATUS				Status Register (In parallel to the SPI command word applied on the MOSI pin, the STATUS register is shifted serially out on the MISO pin)
	Reserved	7	0	R/W	Only '0' allowed
	RX_DR	6	0	R/W	Data Ready RX FIFO interrupt. Asserted when new data arrives RX FIFO[b]. Write 1 to clear bit.
	TX_DS	5	0	R/W	Data Sent TX FIFO interrupt. Asserted when packet transmitted on TX. If AUTO_ACK is activated, this bit is set high only when ACK is received. Write 1 to clear bit.
	MAX_RT	4	0	R/W	Maximum number of TX retransmits interrupt Write 1 to clear bit. If MAX_RT is asserted it must be cleared to enable further communication.
	RX_P_NO	3:1	111	R	Data pipe number for the payload available for reading from RX_FIFO 000-101: Data Pipe Number 110: Not Used 111: RX FIFO Empty
	TX_FULL	0	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
08	OBSERVE_TX				Transmit observe register
	PLOS_CNT	7:4	0	R	Count lost packets. The counter is overflow protected to 15, and discontinues at max until reset. The counter is reset by writing to RF_CH. See page 66 and page 74.
	ARC_CNT	3:0	0	R	Count retransmitted packets. The counter is reset when transmission of a new packet starts. See page 66.
09	CD				
	Reserved	7:1	000000	R	
	CD	0	0	R	Carrier Detect. See page page 74.
0A	RX_ADDR_P0	39:0	0xE7E7E7E7E7	R/W	Receive address data pipe 0. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0B	RX_ADDR_P1	39:0	0xC2C2C2C2C2	R/W	Receive address data pipe 1. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0C	RX_ADDR_P2	7:0	0xC3	R/W	Receive address data pipe 2. Only LSB. MSBytes is equal to RX_ADDR_P1[39:8]
0D	RX_ADDR_P3	7:0	0xC4	R/W	Receive address data pipe 3. Only LSB. MSBytes is equal to RX_ADDR_P1[39:8]
0E	RX_ADDR_P4	7:0	0xC5	R/W	Receive address data pipe 4. Only LSB. MSBytes is equal to RX_ADDR_P1[39:8]
0F	RX_ADDR_P5	7:0	0xC6	R/W	Receive address data pipe 5. Only LSB. MSBytes is equal to RX_ADDR_P1[39:8]

Illustration 23: Registres de configuration du module radio, (datasheet)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
10	TX_ADDR	39:0	0xE7E7E7E7E7	R/W	Transmit address. Used for a PTX device only. (LSByte is written first) Set RX_ADDR_P0 equal to this address to handle automatic acknowledge if this is a PTX device with Enhanced ShockBurst™ enabled. See page 65.
11	RX_PW_P0				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P0	5:0	0	R/W	Number of bytes in RX payload in data pipe 0 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
12	RX_PW_P1				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P1	5:0	0	R/W	Number of bytes in RX payload in data pipe 1 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
13	RX_PW_P2				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P2	5:0	0	R/W	Number of bytes in RX payload in data pipe 2 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
14	RX_PW_P3				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P3	5:0	0	R/W	Number of bytes in RX payload in data pipe 3 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
15	RX_PW_P4				
	Reserved	7:6	00	R/W	Only '00' allowed

Illustration 24: Registres de configuration du module radio, (datasheet)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	RX_PW_P4	5:0	0	R/W	Number of bytes in RX payload in data pipe 4 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
16	RX_PW_P5				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P5	5:0	0	R/W	Number of bytes in RX payload in data pipe 5 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte ... 32 = 32 bytes
17	FIFO_STATUS				FIFO Status Register
	Reserved	7	0	R/W	Only '0' allowed
	TX_REUSE	6	0	R	Reuse last transmitted data packet if set high. The packet is repeatedly retransmitted as long as CE is high. TX_REUSE is set by the SPI command REUSE_TX_PL, and is reset by the SPI commands W_TX_PAYLOAD or FLUSH_TX
	TX_FULL	5	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
	TX_EMPTY	4	1	R	TX FIFO empty flag. 1: TX FIFO empty. 0: Data in TX FIFO.
	Reserved	3:2	00	R/W	Only '00' allowed
	RX_FULL	1	0	R	RX FIFO full flag. 1: RX FIFO full. 0: Available locations in RX FIFO.
	RX_EMPTY	0	1	R	RX FIFO empty flag. 1: RX FIFO empty. 0: Data in RX FIFO.
N/A	ACK_PLD[P]	255:0	X	W	Written by separate SPI command ACK packet payload to data pipe number PPP given in SPI command Used in RX mode only Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled first in first out.
N/A	TX_PLD	255:0	X	W	Written by separate SPI command TX data payload register 1 - 32 bytes. This register is implemented as a FIFO with three levels. Used in TX mode only

Illustration 25: Registres de configuration du module radio, (datasheet)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
N/A	RX_PLD	255:0	X	R	Read by separate SPI command RX data payload register. 1 - 32 bytes. This register is implemented as a FIFO with three levels. All RX channels share the same FIFO
1C	DYNPD[c]				Enable dynamic payload length
	Reserved	7:6	0	R/W	Only '00' allowed
	DPL_P5	5	0	R/W	Enable dyn. payload length data pipe 5. (Requires EN_DPL and ENAA_P5)
	DPL_P4	4	0	R/W	Enable dyn. payload length data pipe 4. (Requires EN_DPL and ENAA_P4)
	DPL_P3	3	0	R/W	Enable dyn. payload length data pipe 3. (Requires EN_DPL and ENAA_P3)
	DPL_P2	2	0	R/W	Enable dyn. payload length data pipe 2. (Requires EN_DPL and ENAA_P2)
	DPL_P1	1	0	R/W	Enable dyn. payload length data pipe 1. (Requires EN_DPL and ENAA_P1)
	DPL_P0	0	0	R/W	Enable dyn. payload length data pipe 0. (Requires EN_DPL and ENAA_P0)
1D	FEATURE[c]			R/W	Feature Register
	Reserved	7:3	0	R/W	Only '00000' allowed
	EN_DPL	2	0	R/W	Enables Dynamic Payload Length
	EN_ACK_PAY[d]	1	0	R/W	Enables Payload with ACK
	EN_DYN_ACK	0	0	R/W	Enables the W_TX_PAYLOAD_NOACK command

Illustration 26: Registres de configuration du module radio, (datasheet)

Annexe 2 : Code source du logiciel créer avec Builder C++

```
#define __WIN__
#include "mysql.h"

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include <time.h>
#include "Unit5.h"
#include "Unit6.h"
#define COEF_JUTANCE_48 18.39
#define COEF_JUTANCE_36 13.55
#define COEF_JUTANCE_24 8.74
#define COEF_JUTANCE_12 3.85

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "CSPIN"
#pragma link "CPort"
#pragma link "CPortCtl"
#pragma resource "*.dfm"
#include <IniFiles.hpp>
TForm1 *Form1;
MYSQL *mySQL;

void Actualisation(void);
void TestKart(int KartAdd, int ItemPos);
void ConnectionMysql(AnsiString Host, AnsiString Utilisateur, AnsiString Pwd,
AnsiString Bdd);
void AjoutMySql(AnsiString Time, AnsiString KartName, AnsiString KartAddr,
AnsiString Tension1, AnsiString Tension2, AnsiString Tension3, AnsiString
Tension4);
int APortee=0;
boolean  FlagMySqlConnect =false;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void DeleteSessions(AnsiString SessionID)
{
 //DELETE FROM `gestionkarting`.`sessionlist` WHERE `sessionlist`.`SessionID` =
0
 //DELETE FROM `gestionkarting`.`tensionkarting` WHERE
`tensionkarting`.`SessionID` = 8
 AnsiString Requete;

Requete.sprintf("DELETE FROM `gestionkarting`.`sessionlist` WHERE
`sessionlist`.`SessionID` = %s",SessionID);
        if (!mysql_query(mySQL,Requete.c_str()))
        {
```

```
            Requete.sprintf("DELETE FROM `gestionkarting`.`tensionkarting` WHERE
`tensionkarting`.`SessionID` = %s",SessionID);
            if (!mysql_query(mySQL,Requete.c_str()))
            {
                ShowMessage("Supprimé avec succes");
            }
            else
            {
               ShowMessage("Aucuns elements suprimer");
            }
        }
        else
        {
          ShowMessage("Erreur avec la requete SQL de suppression de session.");
        }
}
void RefreshSessions(void)
{

  MYSQL_RES *myRES;
  MYSQL_ROW myROW;
  AnsiString aStr,Requete;

  Form6->ComboBox1->Clear();
  Form6->ComboBox2->Clear();

  if (!mysql_query(mySQL, "select * from sessionlist")) {
    myRES = mysql_store_result(mySQL);
    if (myRES) {
        for(unsigned int i = 0; i < myRES->row_count; i++) {
            myROW = mysql_fetch_row(myRES);
            for(unsigned int j = 0; j < mysql_num_fields(myRES); j++) {
                aStr = myROW[j];
                Form6->ComboBox1->Items->Add(aStr);
                Form6->ComboBox2->Items->Add(aStr);
               // ListBox1->Items->Add(aStr);
            }
        }
        mysql_free_result(myRES);
    }
   }
   if (Form6->ComboBox1->Items->Count >0)
   {
       Form6->ComboBox1->ItemIndex = 0;
       Form6->ComboBox2->ItemIndex = 0;
   }
   else
   {
            Requete.sprintf("INSERT INTO sessionlist (SessionID) VALUES('1')");
            if (!mysql_query(mySQL,Requete.c_str()))
            {
                ShowMessage("Aucunes sessions dans la base de données, session 1
ajoutée par defaut!");
                RefreshSessions();
            }
            else
            {
                ShowMessage("Erreur d'acces a la base de données");
```

```
            }

    }
}
void __fastcall TForm1::Apropos1Click(TObject *Sender)
{
Form2->ShowModal();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form3->ShowModal();
}
//---------------------------------------------------------------------------
void AjouteUnKart(AnsiString KartName, AnsiString KartAddr)
{
        Form1->ListKart->Items->Add();

        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add(KartName);
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add(KartAddr);
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add("n/a");
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add("n/a");
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add("n/a");
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add("n/a");
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add("Jamais");
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]->SubItems-
>Add("Jamais");
        Form1->ListKart->Items->Item[Form1->ListKart->Items->Count-1]-
>SubItemImages[7] = 1;

}
void __fastcall TForm1::Button1Click(TObject *Sender)
{
        AjouteUnKart(Edit1->Text,Edit2->Text);
        Edit1->Clear();
        Edit2->Clear();

}
//---------------------------------------------------------------------------
void Actualisation(void)
{
        Form1->Button1->Enabled = false;
        Form1->Button2->Enabled = false;
        Form1->StatusBar1->Panels->Items[1]->Text = "Nombre de karting référencé
: "+IntToStr(Form1->ListKart->Items->Count);
        Form1->ComPort->Connected = true;
        APortee = 0;
        Form1->Memo1->Clear();
        for (int i=0; i<Form1->ListKart->Items->Count;i++)
        {
             TListItem *Item = Form1->ListKart->Items->Item[i];
```

```
            AnsiString Processus = Item->SubItems->Strings[1];
            int Addr;
            Addr = StrToInt(Processus);
            //ShowMessage("Adresse  : "+IntToStr(Addr));
            TestKart(Addr,i);

        }
        Form1->ComPort->Connected = false;
        Form1->StatusBar1->Panels->Items[2]->Text = "Nombre de karting a
portée : "+IntToStr(APortee);
        Form1->Button1->Enabled = true;
        Form1->Button2->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
if ( FlagMySqlConnect == false) ShowMessage("Veuillez activer la connexion a la
base de données MySql avant d'actualiser!");
else        Actualisation();
}

void TestKart(int KartAdd, int ItemPos)
{
    unsigned char Tensions[4];
    unsigned char Message[8];
    int ReTest=3;
    bool FlagReussisOrNot = 0;
    char Heure[100];
    String HeureFormate;
    AnsiString MsgAff;

    time_t timestamp = time(NULL);
    strftime(Heure, sizeof(Heure), "%D\n%T", localtime(&timestamp));

    while (ReTest>0)
    {
            Form1->ComPort->ClearBuffer(true,true);
            Form1->ComPort->TransmitChar(KartAdd);
            //Form1->ComPort->ClearBuffer(true,true);

            while (Form1->ComPort->InputCount() <1);
            Form1->ComPort->Read(Message,1);
            if (Message[0] == '1')
            {
                //ShowMessage("Envoyé");
                Sleep(3000);
                if (Form1->ComPort->InputCount() >=7)
                {

                        Form1->ComPort->Read(Message,7);
                        //  Form1->ComPort->ReadStr(Message,6);
                        if ((Message[0] == '#') && (Message[6] == '$'))
                        {
                                String Test;
//                                Test.sprintf("%02x %02x %02x %02x %02x %02x
%02x",Message[0],Message[1],Message[2],Message[3],Message[4],Message[5],Message[
6]);
```

```
                                MsgAff.sprintf("[%s] Tension du karting 0x%02x
recu",Heure,KartAdd);
                                Form1->Memo1->Lines->Add(MsgAff);
                                Tensions[0] = Message[2];
                                Tensions[1] = Message[3];
                                Tensions[2] = Message[4];
                                Tensions[3] = Message[5];
                                FlagReussisOrNot = 1;
                                ReTest=0;
                        }
                        else
                        {
                                AnsiString Test;
                                Test.sprintf("%02x %02x %02x %02x %02x %02x
%02x",Message[0],Message[1],Message[2],Message[3],Message[4],Message[5],Message[
6]);
                                MsgAff.sprintf("[%s] Tension du karting 0x%02x
non conforme : ",Heure,KartAdd);
                                Form1->Memo1->Lines->Add(MsgAff+Test);
                                if (ReTest==1) FlagReussisOrNot=0;
                                ReTest=ReTest-1;
                        }
                }
                else
                {
                        MsgAff.sprintf("[%s] Tension du karting 0x%02x non
recu",Heure,KartAdd);
                        Form1->Memo1->Lines->Add(MsgAff);
                        if (ReTest==1) FlagReussisOrNot=0;
                        ReTest=ReTest-1;
                }
            } else if (Message[0] == '0')
            {
                MsgAff.sprintf("[%s] Aucuns kart a l'adresse 0x
%02x",Heure,KartAdd);
                Form1->Memo1->Lines->Add(MsgAff);
                if (ReTest==1) FlagReussisOrNot=0;
                ReTest=ReTest-1;
            }

    }
    if (FlagReussisOrNot==1)
    {


        String Jutance;
        Jutance.sprintf("%.2f V",(((Tensions[0]*3.3)/255))*COEF_JUTANCE_48);
        Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[2] = Jutance;
        Jutance.sprintf("%.2f V",(((Tensions[1]*3.3)/255))*COEF_JUTANCE_36);
        Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[3] = Jutance;
        Jutance.sprintf("%.2f V",(((Tensions[2]*3.3)/255))*COEF_JUTANCE_24);
        Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[4] = Jutance;
        Jutance.sprintf("%.2f V",(((Tensions[3]*3.3)/255))*COEF_JUTANCE_12);
        Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[5] = Jutance;
        HeureFormate.sprintf("%s",Heure);
        Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[6] =
HeureFormate;
        Form1->ListKart->Items->Item[ItemPos]->SubItemImages[7]= 0;
```

```
   // printf("%s\n", buffer);
        Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[7] = "Ok";
        APortee = APortee+1;
        AjoutMySql(Heure,Form1->ListKart->Items->Item[ItemPos]->SubItems-
>Strings[0],Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[1],Form1-
>ListKart->Items->Item[ItemPos]->SubItems->Strings[2],Form1->ListKart->Items-
>Item[ItemPos]->SubItems->Strings[3],Form1->ListKart->Items->Item[ItemPos]-
>SubItems->Strings[4],Form1->ListKart->Items->Item[ItemPos]->SubItems-
>Strings[5]);
    }
    else if (FlagReussisOrNot==0)
    {
        Form1->ListKart->Items->Item[ItemPos]->SubItemImages[7]= 2;
        Form1->ListKart->Items->Item[ItemPos]->SubItems->Strings[7] = "Fail";
    }
}
//---------------------------------------------------------------------------



void __fastcall TForm1::Edit1Click(TObject *Sender)
{
Edit1->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit2Click(TObject *Sender)
{
Edit2->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
        Edit1->Clear();
        Edit2->Clear();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Supprimer1Click(TObject *Sender)
{
  if (ListKart->ItemIndex >-1)  ListKart->Items->Item[ListKart->ItemIndex]-
>Delete();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int NbKart,i;
    AnsiString KartName,KartAddr;


    TIniFile *FichierIni = new TIniFile("./ConfigKart.ini");
    NbKart = FichierIni->ReadInteger("Config","NbKart",0);
    for (i=0;i<NbKart;i++)
    {
       KartName.sprintf("KartName_%d",i);
       KartAddr.sprintf("KartAddr_%d",i);
```

```
        AjouteUnKart(FichierIni->ReadString("Config",KartName,"err"),FichierIni-
>ReadString("Config",KartAddr,"err"));
    }
    delete FichierIni;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{

    int NbKart,i;
    AnsiString KartName,KartAddr;

    mysql_close(mySQL);
////////////////////////////////////////////////////////
    TIniFile *FichierIni = new TIniFile("./ConfigKart.ini");
    FichierIni->EraseSection("Config");
    FichierIni->WriteInteger("Config","NbKart",ListKart->Items->Count);

    for (i=0;i<ListKart->Items->Count;i++)
    {
       KartName.sprintf("KartName_%d",i);
       KartAddr.sprintf("KartAddr_%d",i);
       FichierIni->WriteString("Config",KartName,Form1->ListKart->Items-
>Item[i]->SubItems->Strings[0]);
       FichierIni->WriteString("Config",KartAddr,Form1->ListKart->Items-
>Item[i]->SubItems->Strings[1]);
    }
    delete FichierIni;
}

void ConnectionMysql(AnsiString Host, AnsiString Utilisateur, AnsiString Pwd,
AnsiString Bdd)
{
        mySQL = mysql_init(NULL);
        if (mySQL == NULL) ShowMessage("Erreur a la création de l'objet MySql");

        if (!mysql_real_connect(mySQL, Host.c_str(), Utilisateur.c_str(),
Pwd.c_str(), Bdd.c_str(), 0, NULL, 0))
        {
                ShowMessage("Connexion a la base de données MySql impossible,
veuillez vérifier ça configuration en allant dans 'Base de
données>Configuration'");
        }
        else
        {
                ShowMessage("Connexion a la base de données MySql réussie");
        }

}
//---------------------------------------------------------------------------




void __fastcall TForm1::BasededonnesMySql1Click(TObject *Sender)
```

```
{
Form5->ShowModal();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Connexion1Click(TObject *Sender)
{
  TIniFile *FichierIni = new TIniFile("./ConfigKart.ini");
  ConnectionMysql(FichierIni->ReadString("MySql","Host",""),FichierIni-
>ReadString("MySql","User",""),FichierIni-
>ReadString("MySql","Pass",""),FichierIni->ReadString("MySql","BDD",""));
  delete FichierIni;
  FlagMySqlConnect = true;
}
void AjoutSessionMySql(AnsiString SessionID)
{
AnsiString Requete;

Requete.sprintf("INSERT INTO sessionlist (SessionID) VALUES('%s')",SessionID);
        if (!mysql_query(mySQL,Requete.c_str()))
        {

        }
        else
        {
          ShowMessage("Erreur avec la requete SQL d'ajout de session, peut etre
qu'elle existe deja?");
        }
}
void AjoutMySql(AnsiString Time, AnsiString KartName, AnsiString KartAddr,
AnsiString Tension1, AnsiString Tension2, AnsiString Tension3, AnsiString
Tension4)
{
AnsiString Requete,SessionId;
  TIniFile *FichierIni = new TIniFile("./ConfigKart.ini");
  SessionId = FichierIni->ReadString("MySql","SessionID","1");
  delete FichierIni;
Requete.sprintf("INSERT INTO tensionkarting (SessionId, Time,
KartName,KartAddr,Tension1,Tension2,Tension3,Tension4)
VALUES('%s','%s','%s','%s','%s','%s','%s','%s')",SessionId,Time,KartName,KartAdd
r,Tension1,Tension2,Tension3,Tension4);
        if (!mysql_query(mySQL,Requete.c_str()))
        {

        }
        else
        {
          ShowMessage("Erreur avec la requete SQL d'ajout");
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Gestiondessessions1Click(TObject *Sender)
{
if ( FlagMySqlConnect == false) ShowMessage("Veuillez activer la connexion a la
base de données MySql avant d'actualiser!");
else Form6->ShowModal();
}
```

Annexe 3 : Code source du « maître »

```
/*
librairie utilisée :
nrf24l01 lib sample

copyright (c) Davide Gironi, 2012

Released under GPLv3.
Please refer to LICENSE file for licensing information.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

//define debug print enable or disable on uart
#define DEBUGENABLED 1

//include uart
#if DEBUGENABLED == 1
#include "uart/uart.h"
#define UART_BAUD_RATE 2400
#endif

//include nrf24l01
#include "nrf24l01/nrf24l01.h"

//output led
#define LEDOUT_DDR DDRB
#define LEDOUT_PORT PORTB
#define LEDOUT PB2
#define LEDOUTPAUSE 300

static int ModeRechercheOrNot=0; //0 = normale, 1 = Recherche


int main(void) {
    uint8_t i = 0;

    //nrf24l01 variables
    uint8_t bufferout[NRF24L01_PAYLOAD]={0x22,0x22,0x22,0x22,0x22};
    uint8_t bufferout2[NRF24L01_PAYLOAD]={0x50,0x50,0x50,0x50,0x50};
    uint8_t bufferin[NRF24L01_PAYLOAD];

    LEDOUT_DDR |= (1<<LEDOUT); //output
    LEDOUT_PORT &= ~(1<<LEDOUT); //off

    #if DEBUGENABLED == 1
    //init uart
    uart_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );
    #endif

    nrf24l01_MettreAdress(0x30);
```

```
    nrf24l01_init();

    sei();

    LEDOUT_PORT |= (1 << LEDOUT); _delay_ms(LEDOUTPAUSE);
    LEDOUT_PORT &= ~(1 << LEDOUT); _delay_ms(LEDOUTPAUSE);
    LEDOUT_PORT |= (1<<LEDOUT); _delay_ms(LEDOUTPAUSE);
    LEDOUT_PORT &= ~(1<<LEDOUT); _delay_ms(LEDOUTPAUSE);

    for(i=0; i<sizeof(bufferin); i++) bufferin[i] = 0;

    #if DEBUGENABLED == 1 && NRF24L01_PRINTENABLE == 1
        nrf24l01_printinfo(uart_puts, uart_putc);
    #endif

    while(1)
    {
            uint8_t Data;
            Data=uart_getc();

            if (Data!=NULL)
            {
                if (Data == 0xFF)
                {
                    //Rien
                }
                else if (Data == 0xFE)
                {
                    //Mode Recherche ON
                    ModeRechercheOrNot = 1;
                    uart_putc('1');
                }
                else if (Data == 0xFD)
                {
                    //Mode Recherche OFF
                    ModeRechercheOrNot = 0;
                    uart_putc('0');
                }
                else
                {
                    if (ModeRechercheOrNot == 1) // si on est en mode recherche
on envoie une mauvaise trame pour pas avoir de retour mais juste recup l'etat
d'envoie
                    {
                        uint8_t addrBase[NRF24L01_ADDRSIZE] ={0x20,0x20,
0x20,0x20, 0x20}  ;
                        addrBase[4] = Data;
                        nrf24l01_settxaddr(addrBase); // Met l'addresse du
module
                        uint8_t writeret = nrf24l01_write(bufferout2);
                        if(writeret == 1)
                            uart_putc('1');
                        else
                            uart_putc('0');

                    }
                    else if (ModeRechercheOrNot == 0) // Mode normal, on envoie
la bonne trame pour recevoir l'info
```

```
                    {
                        uint8_t addrBase[NRF24L01_ADDRSIZE] ={0x20,0x20,
0x20,0x20, 0x20}  ;
                        addrBase[4] = Data;
                        nrf24l01_settxaddr(addrBase); // Met l'addresse du
module
                        uint8_t writeret = nrf24l01_write(bufferout);
                        if(writeret == 1)
                            uart_putc('1');
                        else
                            uart_putc('0');
                    }

                }
            }

            uint8_t pipe = 0;
            if(nrf24l01_readready(&pipe))
            {
                    LEDOUT_PORT |= (1<<LEDOUT); _delay_ms(LEDOUTPAUSE);
                    LEDOUT_PORT &= ~(1<<LEDOUT); _delay_ms(LEDOUTPAUSE);
                    //uart_puts("reception :");
                    nrf24l01_read(bufferin);
                    if (bufferin[0] != '@' )uart_putc('#');
                    for (i=0;i<5;i++)
                    {
                       uart_putc(bufferin[i]);
                    }
                    if (bufferin[0] != '@' )uart_putc('$');
            }
    }
}
```

Annexe 4 : Code source de « l'esclave »

```
/*
librairie utilisée
nrf24l01 lib sample

copyright (c) Davide Gironi, 2012

Released under GPLv3.
Please refer to LICENSE file for licensing information.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

//define debug print enable or disable on uart
#define DEBUGENABLED 1

//include uart
#if DEBUGENABLED == 1
#include "uart/uart.h"
#define UART_BAUD_RATE 2400
#endif

//include nrf24l01
#include "nrf24l01/nrf24l01.h"

//output led
#define LEDOUT_DDR DDRB
#define LEDOUT_PORT PORTB
#define LEDOUT PB2
#define LEDOUTPAUSE 300

unsigned int LitTension(uint8_t LePortADC);
uint8_t LireAdresse(void);
uint8_t AdresseLue;



//main here
int main(void) {
    uint8_t i = 0;

    //nrf24l01 variables
    uint8_t bufferout[NRF24L01_PAYLOAD];
    uint8_t bufferin[NRF24L01_PAYLOAD];

    LEDOUT_DDR |= (1<<LEDOUT); //output
    LEDOUT_PORT &= ~(1<<LEDOUT); //off

    #if DEBUGENABLED == 1
    //init uart
    uart_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );
```

```
    #endif
     AdresseLue = LireAdresse();
     //AdresseLue = 0x20;

                #if DEBUGENABLED == 1
                char pipebuffer[5];
                uart_puts("Adresse : ");

                    sprintf(pipebuffer,"%02X",AdresseLue);
                    uart_puts(pipebuffer);

                uart_puts("\r\n");
                #endif
   //
    nrf24l01_MettreAdress(AdresseLue);
    nrf24l01_init();
    nrf24l01_flushRXfifo();
    sei();
    LEDOUT_PORT |= (1 << LEDOUT); _delay_ms(LEDOUTPAUSE);
    LEDOUT_PORT &= ~(1 << LEDOUT); _delay_ms(LEDOUTPAUSE);
    LEDOUT_PORT |= (1<<LEDOUT); _delay_ms(LEDOUTPAUSE);
    LEDOUT_PORT &= ~(1<<LEDOUT); _delay_ms(LEDOUTPAUSE);

    for(i=0; i<sizeof(bufferin); i++) bufferin[i] = 0;
    #if DEBUGENABLED == 1 && NRF24L01_PRINTENABLE == 1
        nrf24l01_printinfo(uart_puts, uart_putc);
    #endif

    DDRC &= 0xD0; //pour foutre en entrée PC5 tout en les foutant en pullup
    PORTC |= 0x20;
    while(1)
    {
            uint8_t pipe = 0;
            if(nrf24l01_readready(&pipe))
            {
                #if DEBUGENABLED == 1
                uart_puts("Receptions de donnes, trame :\r\n");
                #endif

                nrf24l01_read(bufferin);

                if (
(bufferin[0]==0x22)&&(bufferin[1]==0x22)&&(bufferin[2]==0x22)&&(bufferin[3]==0x2
2)&&(bufferin[4]==0x22) )
                {
                    #if DEBUGENABLED == 1
                    uart_puts("Demande d'actualisation recu\r\n");
                    #endif
                    bufferout[0]=AdresseLue;
                    bufferout[1]=LitTension(3);
                    bufferout[2]=LitTension(2);
                    bufferout[3]=LitTension(1);
                    bufferout[4]=LitTension(0);

                    _delay_ms(2000); //Le temps que l'autre module repasse dans
le mode recepteur ..
                    #if DEBUGENABLED == 1
                    uart_puts("Envoie des donnes...");
```

```
                    uint8_t addrBase[NRF24L01_ADDRSIZE] = {0x20, 0x20, 0x20,
0x20, 0x30};
                    nrf24l01_settxaddr(addrBase); // Met l'addresse du maitre
pour lui envoyer les infos
                    #endif
                  // nrf24l01_MettreAdress(0x30);
                    uint8_t writeret = nrf24l01_write(bufferout);
                 //   nrf24l01_settxaddr(addrtx0);
                    #if DEBUGENABLED == 1
                    if(writeret == 1)
                        uart_puts("ok\r\n");
                    else
                        uart_puts("failed\r\n");
                    #endif

                    LEDOUT_PORT |= (1<<LEDOUT); _delay_ms(LEDOUTPAUSE);
                    LEDOUT_PORT &= ~(1<<LEDOUT); _delay_ms(LEDOUTPAUSE);
                    #if DEBUGENABLED == 1
                    _delay_ms(1000);
                    #endif
                    /// Passe en role emeteur, lit les tensions, les met dans
buffer, envoie les tensions, repasse en mode recepteur
                }
            }
            _delay_ms(10);
    }
}

uint8_t LireAdresse(void)
{
   uint8_t Valeur;
   DDRD   &= 0x03;
   PORTD  |=  0xFC;
   DDRB   &= 0x3F;
   PORTB  |= 0xC0; // pullup
   Valeur = 0;
   Valeur += ((PIND<<5)&0x80);
   Valeur += ((PIND<<3)&0x40);
   Valeur += ((PIND<<1)&0x20);
   Valeur += ((PINB>>2)&0x10);
   Valeur += ((PINB>>4)&0x08);
   Valeur += ((PIND>>3)&0x04);
   Valeur += ((PIND>>5)&0x02);
   Valeur += ((PIND>>7)&0x01);
   return (Valeur&0xFF);
}

unsigned int LitTension(uint8_t LePortADC)
{
    ADMUX = (1 << REFS0) | (1 << ADLAR) | (LePortADC);
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADSC);
    while (ADCSRA & (1 << ADSC)) continue;
    _delay_us(10);
    return ADCH;
}
```

Index des illustrations

Bibliographie

[1] **atmel**. *www.atmel.com*, 2013, [En ligne]. (Page consultée le 15/10/2013) <www.atmel.com>
[2] **sitelec**. *sitelec.org*, , [En ligne]. (Page consultée le 15/10/2013) <http://sitelec.org>
[3] **astuces-pratiques**. *astuces-pratiques*, , [En ligne]. (Page consultée le 15/10/2013) <http://astuces-pratiques.fr>
[4] **gizmosnack**. *gizmosnack.blogspot.fr*, , [En ligne]. (Page consultée le 15/10/2013) <http://gizmosnack.blogspot.fr>
[5] **e-kart**. *e-kart*, , [En ligne]. (Page consultée le 15/10/2013) <http://e-kart.com>