



Ballon Okour Supélec Espace 2003-2004



Claire Cizaire
Nicolas Couronneau
Aline Meuris
Benoît Legrand

Sommaire

<u>1. Présentation générale.....</u>	<u>3</u>
<u>2. Expériences embarquées.....</u>	<u>4</u>
2.1. Synoptique.....	4
2.2. Mesure de radioactivité.....	4
2.3. Mesures de température et pression.....	9
2.4. Localisation GPS.....	13
2.5. Expérience GSM.....	14
<u>3. Alimentations.....</u>	<u>23</u>
3.1. Présentation.....	23
3.2. Bilan électrique.....	23
3.3. Synthèse.....	23
3.4. Conception.....	24
3.5. Réalisation.....	25
3.6. Bilan de puissance	25
<u>4. Télémessures.....</u>	<u>27</u>
4.1. Emission.....	27
4.2. Modulation FSK.....	27
4.3. Trames télémessures.....	28
4.4. Réception.....	30
<u>5. Lâcher du ballon.....</u>	<u>31</u>
5.1. Cadre du lancement.....	31
5.2. Récupération du ballon.....	31
5.3. Récupération des résultats.....	32
<u>6. Exploitation des résultats.....</u>	<u>33</u>
6.1. Traitement des données.....	33
6.2. Données générales sur le vol.....	33
6.3. Atmosphère.....	34
6.4. Trajectoires.....	39
<u>7. Bilan.....</u>	<u>41</u>
7.1. Organisation du projet.....	41
7.2. Résultats.....	41
7.3. Conclusion.....	41
<u>8. Remerciements.....</u>	<u>42</u>
<u>9. Annexes.....</u>	<u>42</u>
9.1. Expérience GSM.....	42
9.2. Carte principale - Télémessures.....	51
9.3. Alimentations.....	62

Supélec Espace

Supélec - Plateau du Moulon - 91192 Gif-sur-Yvette Cedex

espace@supélec.fr

Association déclarée à la préfecture de l'Essonne – n°310242

1. Présentation générale

Le projet de ballon stratosphérique Okour de Supélec Espace, association de l'école Supélec (campus de Gif sur Yvette en Essonne), fut réalisé par 4 étudiants durant l'année 2003-2004. Les idées motivant sa réalisation étaient l'utilisation d'un GPS pour suivre le ballon, et donc le récupérer, et la mesure de la radioactivité en altitude.

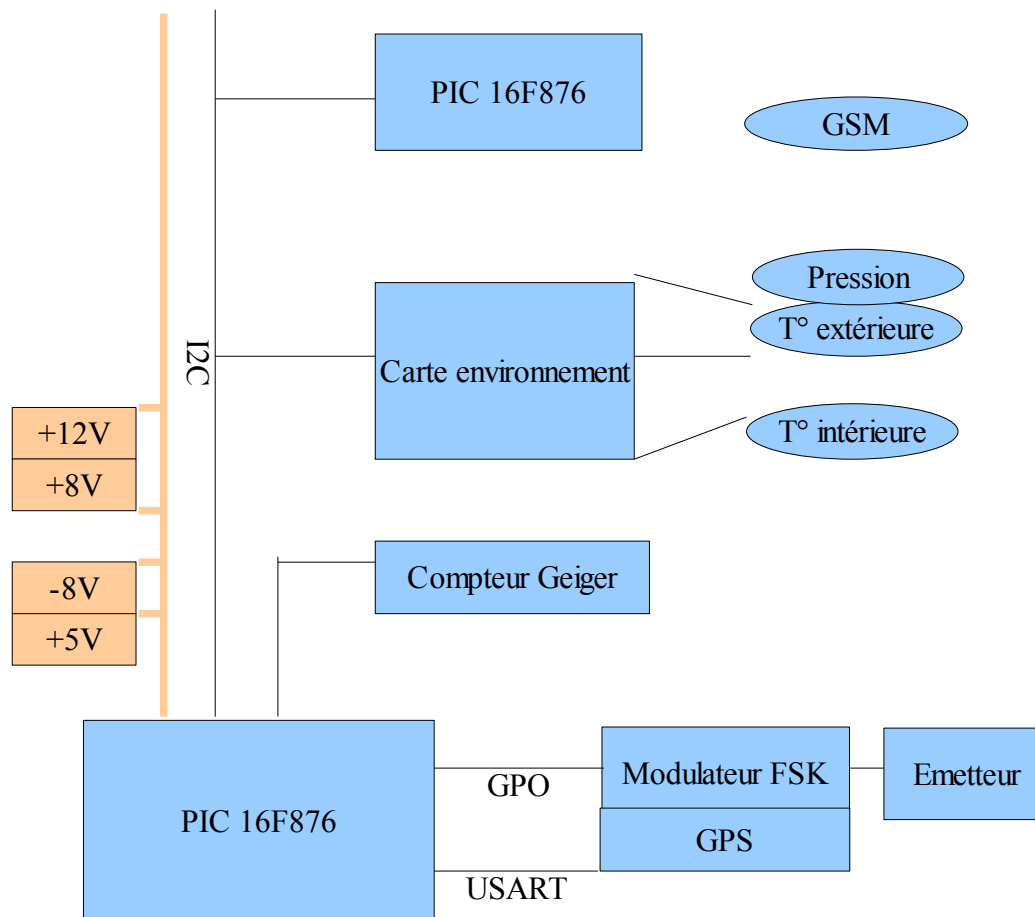
Commencé en octobre, le projet a été réalisé au sein de l'association, en dehors de la scolarité, et avec des moyens techniques et financiers assez limités. Pour pallier au manque de matériel, nous avons largement utilisé la base technique de l'association nationale Planète Sciences.

Après un premier report, le ballon fut finalement lâché le 19 septembre 2004 depuis Buthiers (77), suivi par deux voitures et retrouvé quelques minutes après son atterrissage. Malgré quelques problèmes avec la réception des télémesures, les résultats des expériences ont pu être exploités avec intérêt.

Ce projet a été doublement couronné de succès, car en plus de la réussite du vol, il fut sélectionné par l'ESA pour le programme étudiant du 24ème Congrès International d'Astronautique (IAC 2004). L'un d'entre nous a ainsi pu partir présenter le projet à Vancouver (Canada), en octobre.

2. Expériences embarquées

2.1. Synoptique



2.2. Mesure de radioactivité

2.2.1. Présentation

L'expérience principale consiste à mesurer la radioactivité en altitude, ce qui présente plusieurs difficultés :

- Il est très difficile de trouver des prédictions pour les altitudes des ballons. En dessous de 10km, on peut trouver des études concernant l'exposition aux rayonnements des personnels navigants, et au dessus de 300km on peut trouver des résultats d'expériences embarquées à bord de satellites.
- Dans les quelques documents présentant des résultats à nos altitudes, les grandeurs étudiées ne sont pas mesurables dans un ballon, en raison de l'équipement nécessaire. On trouve des mesures d'intensité du rayonnement en fonction de son énergie, du flux de certaines particules

données, etc. Les différentes énergies des particules compliquent l'estimation du débit de dose qui serait donné par un compteur Geiger classique.

- Divers capteurs sont possibles, mais les contraintes de masses et de coût limitent énormément les possibilités. Après quelques échanges intéressants avec l'IRSN (Institut de Recherche en Sécurité Nucléaire), il s'est avéré que la mesure la plus intéressante est le LET : Transfert d'Énergie Linéique, qui permet de mesurer l'énergie des rayonnements. Cependant, ces capteurs sont assez coûteux et le traitement du signal qui doit suivre est assez complexe (échantillonnage de l'impulsion de la particule pour en mesurer l'énergie).

En définitive, nous avons fait au plus simple en choisissant un compteur Geiger du commerce, ce qui permet de mesurer le flux de particules d'une certaine énergie dans le tube. Le compteur, choisi pour son coût, sa simplicité et sa disponibilité, est un modèle Quartex RD8901.

2.2.2. Modification du compteur

On souhaite récupérer les impulsions en sortie du tube, ce qui demande d'étudier un peu le fonctionnement du compteur. A l'oscilloscope, on observe que les impulsions en sortie du tube se présentent comme des pics d'environ 50 ns, et de tension maximale 300V. La mesure de tensions n'a ici pas beaucoup de signification, car les charges transférées sont extrêmement faibles, et le courant maximal est certainement de quelques nanoampères. Le fabricant relie ce signal à l'entrée d'une bascule RS par l'intermédiaire d'une résistance de 1M Ω . Lorsque qu'une particule traverse le tube, l'impulsion met à 1 la sortie de la bascule reliée à l'entrée du microcontrôleur du compteur (un Atmel). Le microcontrôleur acquitte alors la nouvelle impulsion en remettant à zéro la sortie de la bascule (mécanisme d'interruption externe). On utilisera un montage similaire, en récupérant le signal en sortie du tube et en l'injectant sur une entrée d'interruption du microcontrôleur PIC. Entre les deux, on placera toutefois une porte en logique rapide et CMOS (la commutation doit se faire avec un courant quasi-nul, seules quelques charges sont transférées en sortie du tube), qui fera office de « fusible » pour protéger l'entrée du PIC. Ce montage n'est pas très propre mais fonctionne très bien, avec le composant (74hc85) pistocollé sur le circuit d'origine.

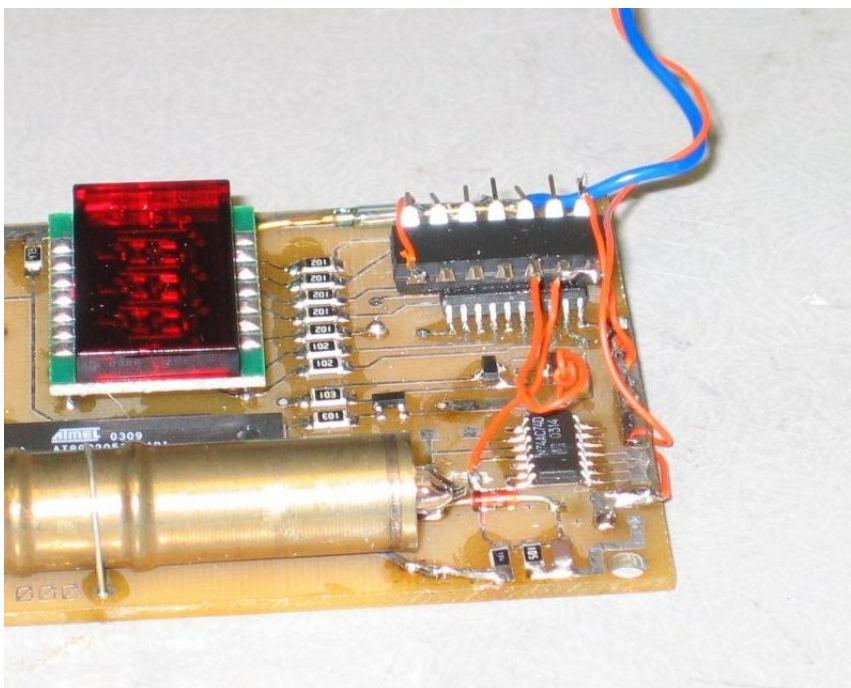


Illustration 1: Vue du compteur Geiger démonté

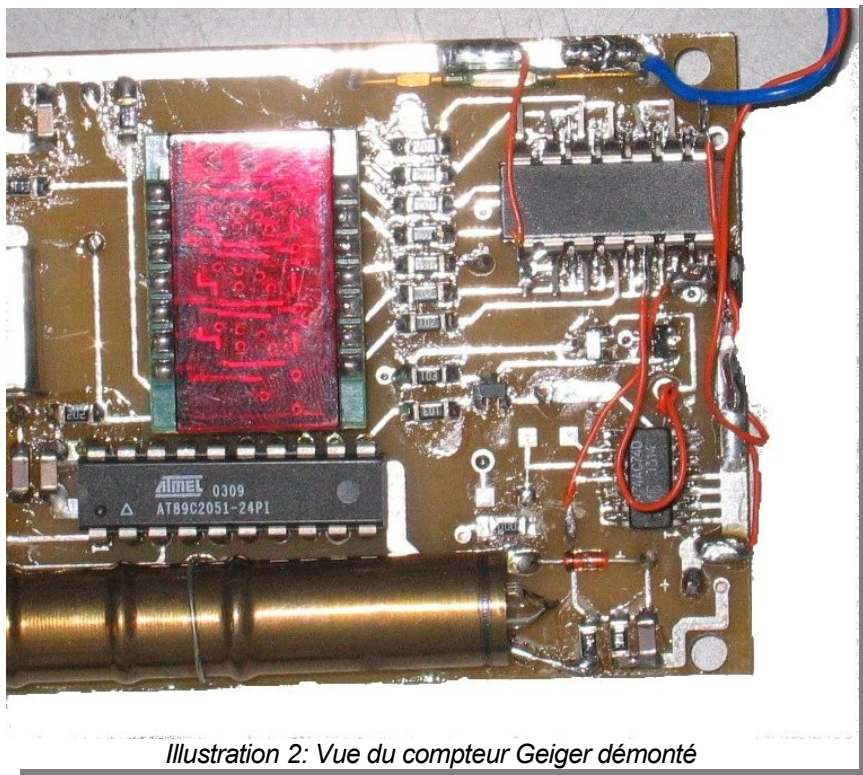


Illustration 2: Vue du compteur Geiger démonté

2.2.3. Calibration du compteur

Nous avons eu accès à un compteur professionnel de référence et à deux sources radioactives d'énergies différentes : Cobalt 60 et Sodium 22. L'objectif était de trouver la relation entre le nombre d'impulsions et le débit de dose mesuré.

D'après la notice du Quartex RD8901, la gamme des énergies mesurables est de :

- 100 keV à 1,2 MeV pour les particules beta
- 350 keV à 1,2 MeV pour les gamma

Sensibilité du tube : 78 impulsions / μ Rem (donc : 78 cps = 1 μ Rem / s = 3600 μ Rem / h = 36 μ Sv / h soit 0,46 (μ Sv/h) / cps)

2.2.3.1. Déroulement

On place la source à égale distance des deux compteurs. Comme le rayonnement est isotrope, les deux compteurs doivent recevoir la même dose. On intègre les mesures sur 3 minutes pour le Quartex (en modifiant le programme de vol pour compter 3 minutes d'impulsions), le Radiagem intégrant sur quelques dizaines de secondes (mais on ne note aucune fluctuation sur le radiagem).

On renouvelle 3 fois chaque mesure avant de passer à une autre distance.

2.2.3.2. Calibration Cobalt 60 (Gamma à 1,1 et 1,3 MeV)

Distance (cm)	10	7	5	4	3	2
Mesure Gamma (μ Sv/h)	1,90	3,70	6,30	6,90	10,60	15,50
Moyenne (cps)	2,67	7,65	12,54	14,47	20,64	30,17
Variance	20,23	50,82	84,56	74,29	102,99	158,56
Ecart type	4,50	7,13	9,20	8,62	10,15	12,59

Illustration 3: Relevés bruts calibration Cobalt 60
6/63

Calibration using Cobalt 60

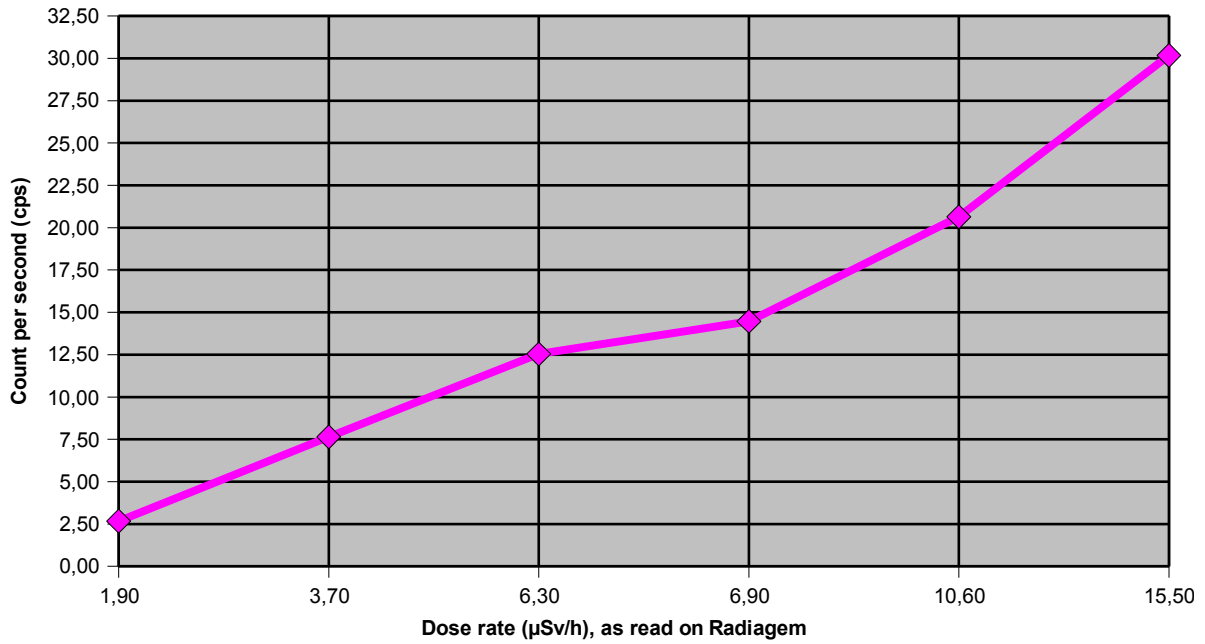


Illustration 4: Graphe relevés bruts calibration Cobalt 60

Calibration Sodium 22 (beta à 550 keV et gamma à 1,2MeV)

Distance (cm)	10,5	7	5	4	3	2	1
Mesure Gamme (µSv/h)	1,3	3,1	6,5	6,9	10,3	16,3	32
Moyenne	2,18	4,21	8,09	11,56	14,48	28,42	88,49
Variance	14,33	28,23	25,43	51,66	83,77	109,22	342,88
Ecart type	3,79	5,31	5,04	7,19	9,15	10,45	18,52

Illustration 5: Relevés bruts calibrations Sodium 22

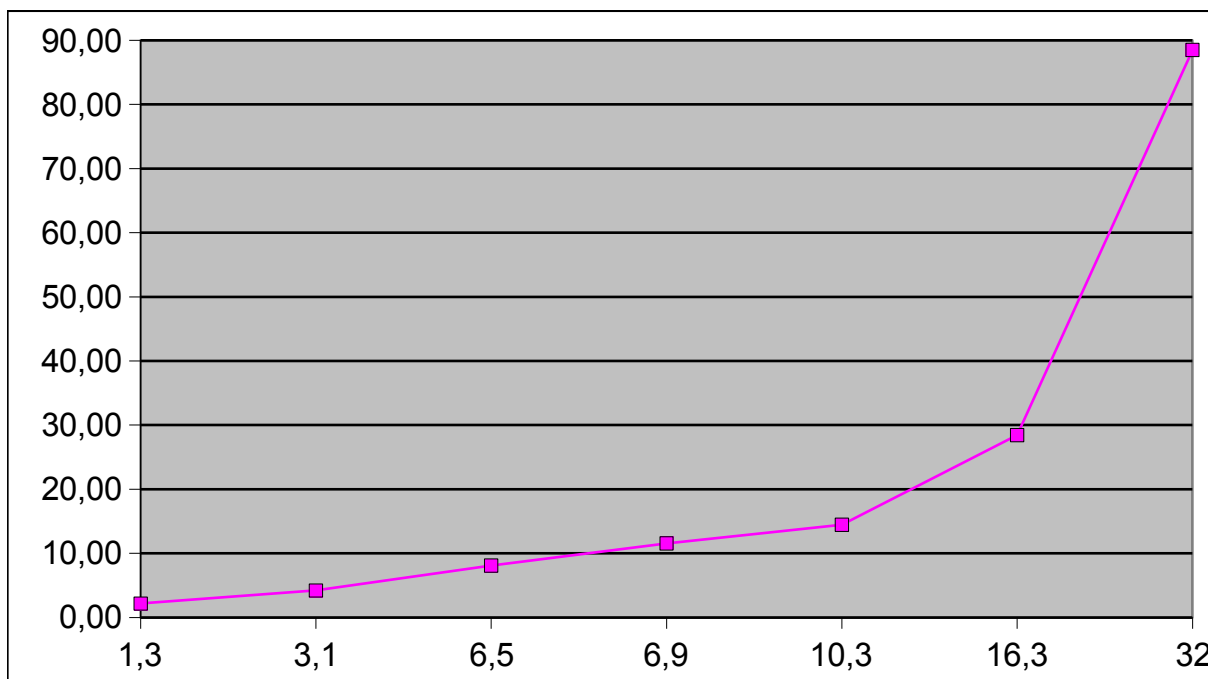


Illustration 6: Graphe relevés bruts calibration Sodium 22 (μSv/h – cm)

A l'issue de la calibration, on note une correspondance linéaire entre le Radiagem (compte de référence) et le Quartex RD8901 embarqué dans le ballon, pour une mesure de particules gamma.

La calibration avec la source Sodium semble montrer une forte sensibilité aux radiations de basses énergies à débit élevé, par rapport à des particules d'énergie plus élevées même de même flux.

Régressions linéaires :

- Cobalt 60 : $(0,5 \times (\text{cps}) + 0,07) (\mu\text{Sv/h})$
- Sodium 22 : $(0,34 \times (\text{cps}) + 3,29) (\mu\text{Sv/h})$

Au final, on prendra la relation annoncée dans la notice qui se trouve être bien vérifiée avec une marge de 10 % sur la calibration au Cobalt.

2.3. Mesures de température et pression

Nous avons choisi de réaliser une carte analogique regroupant des mesures de température à l'intérieur et à l'extérieur de la nacelle, ainsi qu'une mesure de pression. Face aux difficultés lourdes rencontrées au moment des tests de télémétrie, nous avons opté en dernier recours pour la programmation de deux capteurs numériques de température et nous avons supprimé la mesure de pression.

2.3.1. Prédéterminations

- Température atmosphérique de 0 à 40 000 m comprise entre -60°C et 35°C
- Température à l'intérieur de la nacelle comprise entre -10°C et 30°C .
- Pression atmosphérique de 0 à 40 000 m comprise entre 5 kPa et 101 kPa
- Tensions de sortie comprises entre 0 et 5V.

2.3.2. Carte analogique

Le capteur analogique de température utilisé est le AD590, qui délivre un courant dont la valeur en μA correspond à la température en Kelvin. Des simulations sur P-Spice ont permis d'établir la valeur des résistances de gain et d'offset pour sortir une tension entre 0 et 5V. Un étalonnage a été réalisé dans l'étuve de la base technique de Ris-Orangis (pouvant fournir des températures de -80°C à 80°C) pour régler avec précision les résistances ajustables.

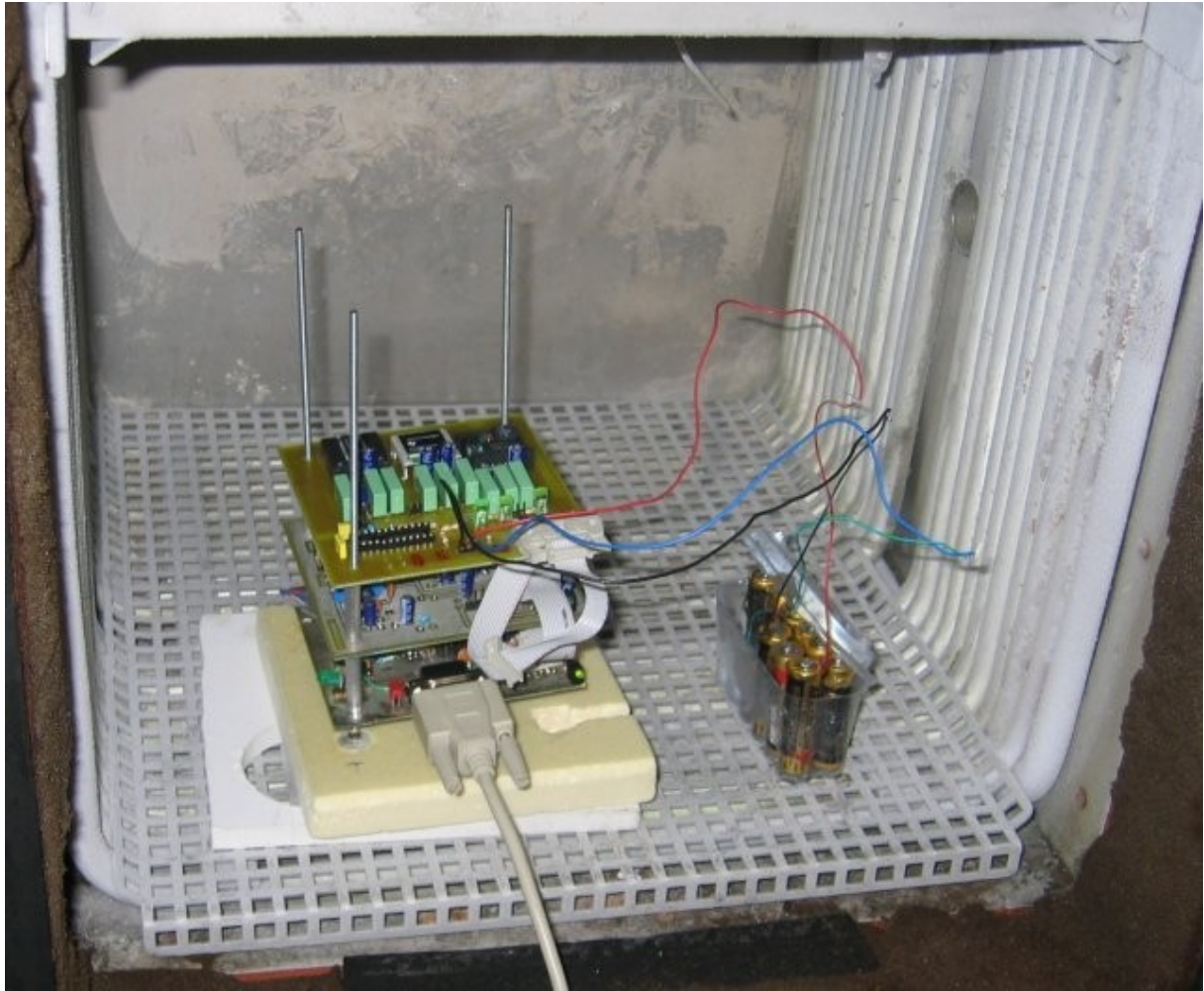
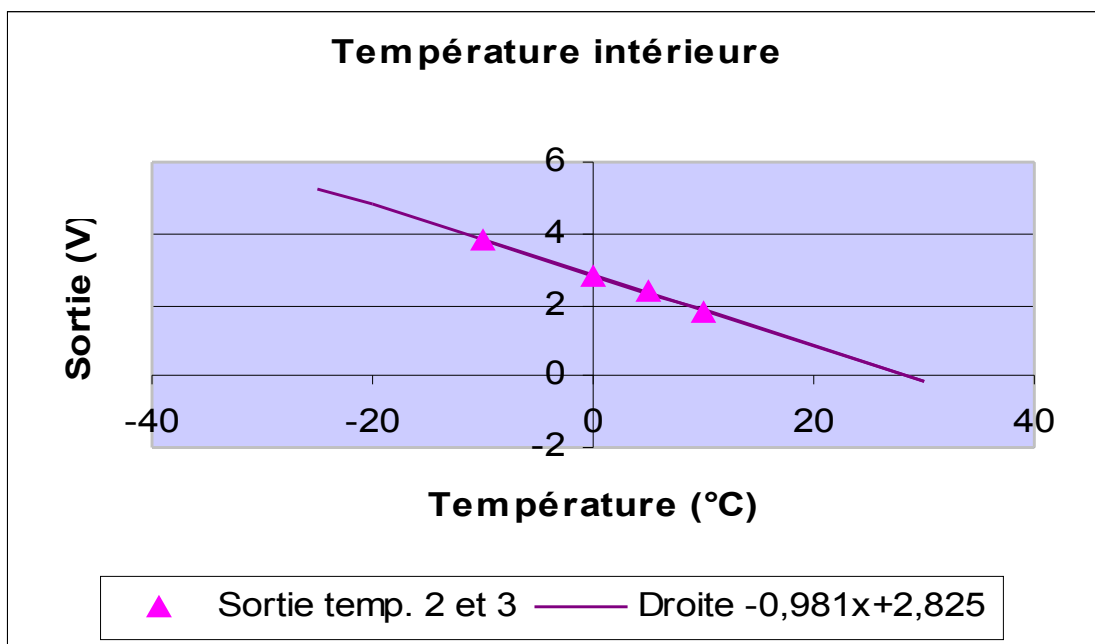
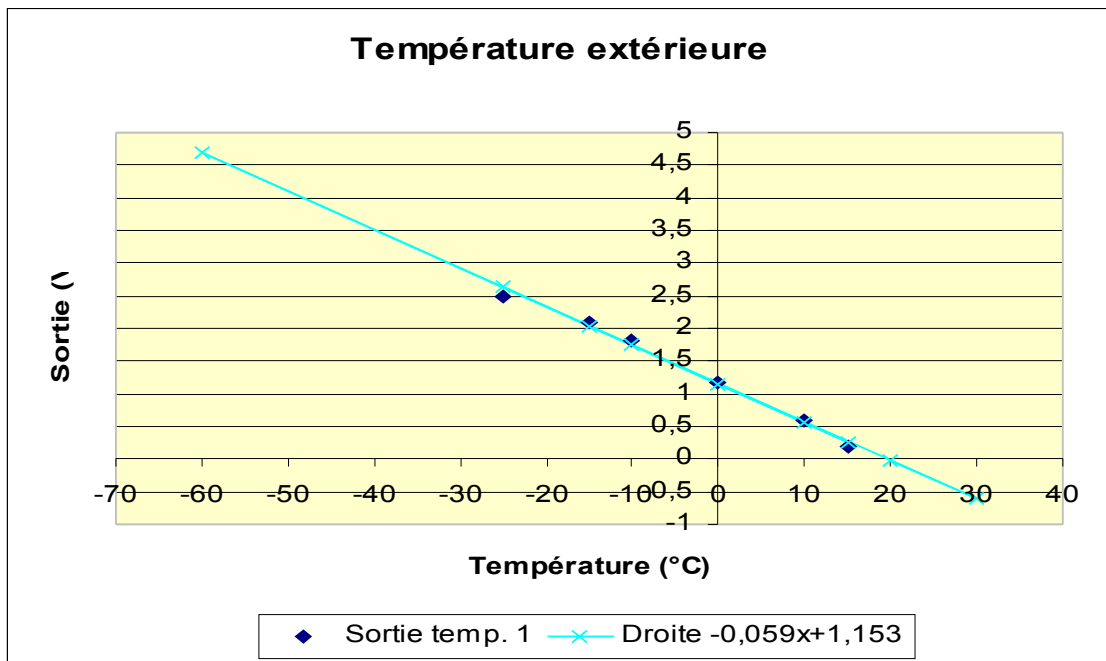


Illustration 7: Cartes électroniques en étuve, sans le ballon

2.3.2.1. Schéma de la carte analogique

(en annexe)

2.3.2.2. Résultats de l'étalonnage



2.3.2.3. Problèmes de CEM

Une chaîne de télémessure complète a été mise en œuvre pour observer les résultats reçus au sol. Le fonctionnement de l'émetteur à proximité de la carte analogique perturbait totalement les mesures de température. On observait des fluctuations de l'ordre du volt, des saturations subites des capteurs... Nous avons isolé au maximum la carte de l'émetteur (boîte d'aluminium, éloignement des capteurs, blindage des câbles, plan de masse...) pour limiter les courants parasites. Face au manque de fiabilité des résultats, nous avons remplacé toute cette carte par des capteurs numériques.

2.3.3. Programmation de capteurs numériques

Nous avons embarqué deux capteurs TMP100, un pour mesurer la température intérieure, un pour la température extérieure.

```
* Driver pour le capteur de température TMP100

#ifndef __TMP100_C__
#define __TMP100_C__
#include "ledvie.c"

/**
 * Retourne la température lue par le capteur <code>n</code>
 *
 * @param numéro du capteur (<b>pas</b> adresse I2C).
 * @return température en degrés, signed int16
 */
int16
lireTmp100 (int8 n)
{
    int16 val;
    int k = 0;

    restart_wdt ();
    // Start conversion (W)
    // configuration du capteur
    i2c_start ();
    // adresse + test
    if(i2c_write (0x90 | (n << 1)))
    {
        // Problème I2C : esclave inaccessible
        for (k = 0; k < 10; k++)
        {
            delay_ms (200);
            changerLed ();
        }
    }
    // pointeur sur registre de configuration
    i2c_write (0x01);
    // registre de configuration
    i2c_write (0);
    i2c_stop ();

    //lecture de la température
    i2c_start ();
    // adresse
    i2c_write (0x90 | (n << 1));
    // pointeur sur registre de température
    i2c_write (0);

    i2c_start ();
    i2c_write (0x90 | (n << 1) | 1);

    val = (i2c_read () << 8);
    val += i2c_read (0);
    i2c_stop ();

    return val;
}

#endif // __TMP100_C__
```

2.4. Localisation GPS

2.4.1. Présentation



On utilise un récepteur Motorola Oncore VP, à 8 canaux parallèles, qui s'interface simplement avec une liaison série, au format RS232 et aux niveaux TTL. Ce modèle est disponible à petit prix comme fin de stock.

Illustration 8: Motorola Oncore VP

L'alimentation se fait sous 5V et 250mA, avec l'antenne amplifiée. Cette consommation est assez importante, surtout pour un ballon. Les nouveaux récepteurs (M12, Laipac), fonctionnent en 3,3V et consomment beaucoup moins. Ils sont très intéressants pour des ballons.

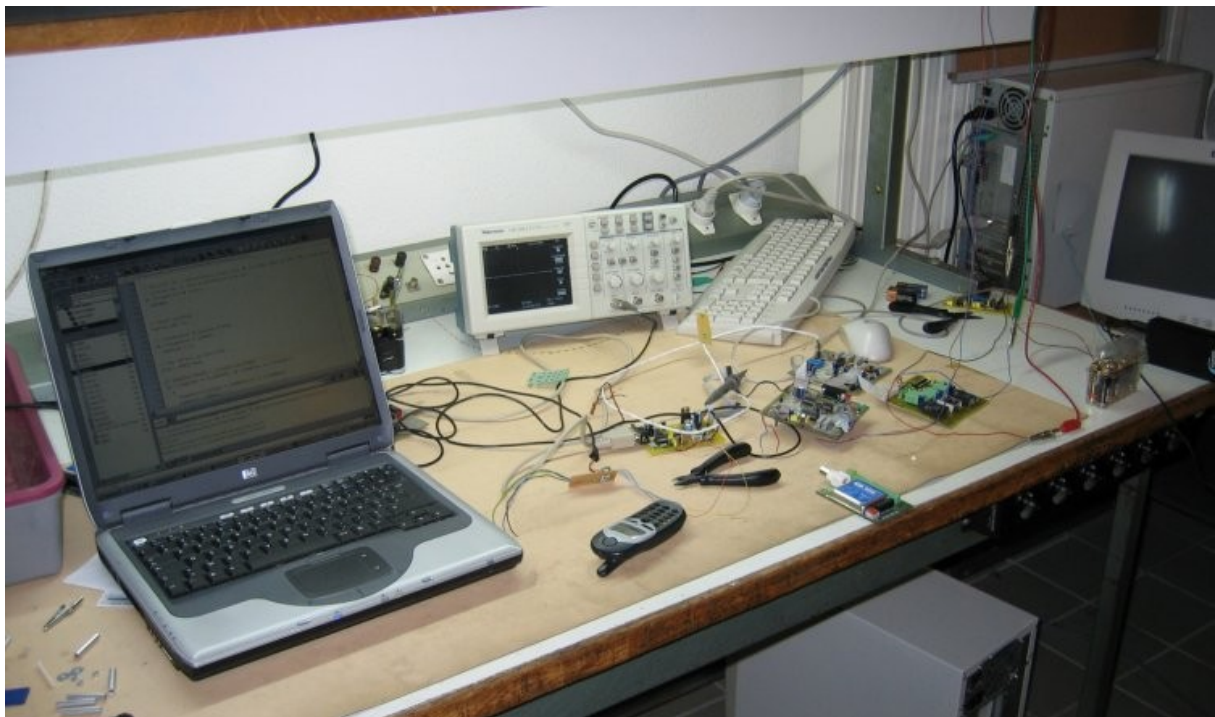


Illustration 9: Plan de travail pour le développement de l'interface GSM-GPS (à la base technique de Ris-Orangis)

2.4.2. Position de l'antenne GPS

Pour faire taire certaines craintes, nous n'avons rencontré aucun problème d'interférence entre l'émetteur Kiwi et le GPS. L'antenne (modèle patch amplifié) est fixée à même le ballon, avec une plaque de cuivre comme plan de masse.

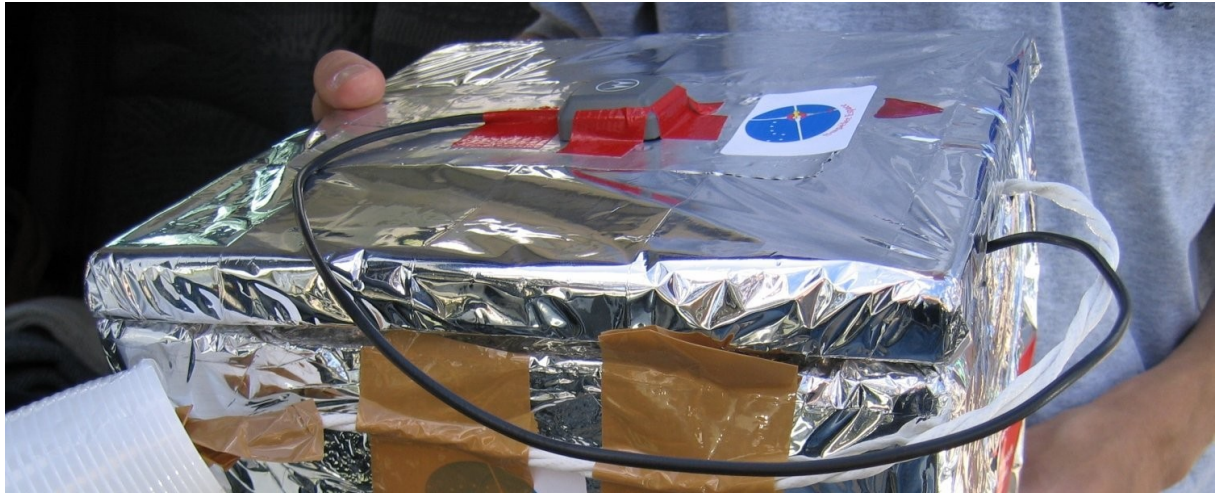


Illustration 10: Placement antenne GPS, sur le haut de la nacelle

Les problèmes déjà rencontrés entre un module GPS et l'émetteur sont peut-être dus à des interférences couplage sur les alimentations, plutôt que par rayonnement sur les étages d'amplification du GPS. Ici nous utilisons deux blocs de piles distincts pour l'émetteur et pour le GPS. Nous n'avons rencontré aucun problème avec l'utilisation d'une alimentation à découpage (filtrée avec des filtres LC en pi et des ferrites) pour le GPS.

2.5. Expérience GSM

Note : Cette partie reprend une étude réalisée par le binôme Claire-Nicolas dans le cadre d'un projet scolaire de 1ère année.

2.5.1. Présentation

La liaison radio de la télémétrie est souvent interrompue à proximité du sol, suite à la présence d'obstacles (forêt, immeubles) ou de dommages lors de l'atterrissage.

Comme nous ne souhaitons pas perdre le matériel embarqué, nous voulons connaître avec précision le point de chute du ballon. Nous utiliserons un téléphone GSM pour envoyer les dernières positions du ballon. On considère en effet que le réseau GSM est plus fiable que notre télémétrie.

Notre projet consiste à récupérer les données de position du GPS, les envoyer par SMS à un portable au sol, via un autre portable embarqué dans le ballon. Le message devra être répété à intervalles réguliers sur le dernier kilomètre pour être sûr de la présence du réseau.

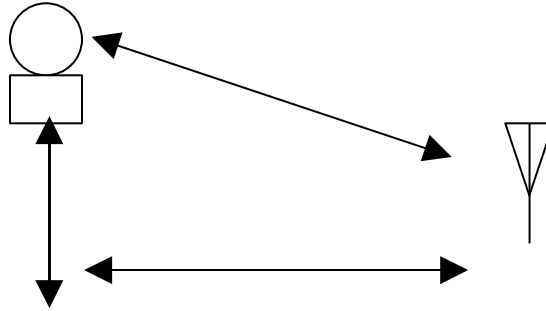
2.5.2. Cahier des charges

2.5.2.1. Fréquence d'envoi

L'envoi des SMS n'est prévu que durant la dernière partie de la descente. On peut aussi envisager l'envoi d'un SMS de bon fonctionnement lors de l'ascension. Il reste à définir le nombre d'envois et l'altitude à partir de laquelle on peut espérer une bonne réception. L'absence de documentation à ce sujet est problématique.

En utilisant les documentations des fabricants d'antennes, on a une idée du diagramme de rayonnement, très concentré dans un plan horizontal. De plus les opérateurs inclinent souvent les antennes (tilt) vers le bas pour améliorer les transmissions. Pour une ouverture de l'antenne de la station de base de 8° avec un tilt de 6° , l'atténuation est de -3db à 2° au dessus de l'horizon.

On peut se fixer 5° pour la réception, et avec une portée optique de 30km (recommandation de portée de la norme GSM), l'altitude maximale de réception est :



$$H = 30 \cdot \sin(5^\circ)$$

$$H = 2,6 \text{ km.}$$

L'idéal serait d'obtenir une dizaine de messages jusqu'à l'atterrissage, donc un message tous les 250m.

La vitesse de descente du ballon, imposée par le parachute CNES, est de 5m/s. Il faudrait donc transmettre toutes les 50s.

A priori, on peut se permettre une fréquence d'envoi plus élevée, la durée d'un envoi étant de quelques secondes. Un message tous les 100m serait bien.

2.5.2.2. Contenu du message

Ce sont les données essentielles à la localisation, en provenance du GPS.

Un seul message SMS ne peut dépasser 160 caractères.

dd°mm.mmmm (N/S)	ddd°mm.mmmm (W/E)	aaaaaM	hh :mm :ss
altitude	longitude	altitude	heure

Taille : 40 caractères (espaces compris).

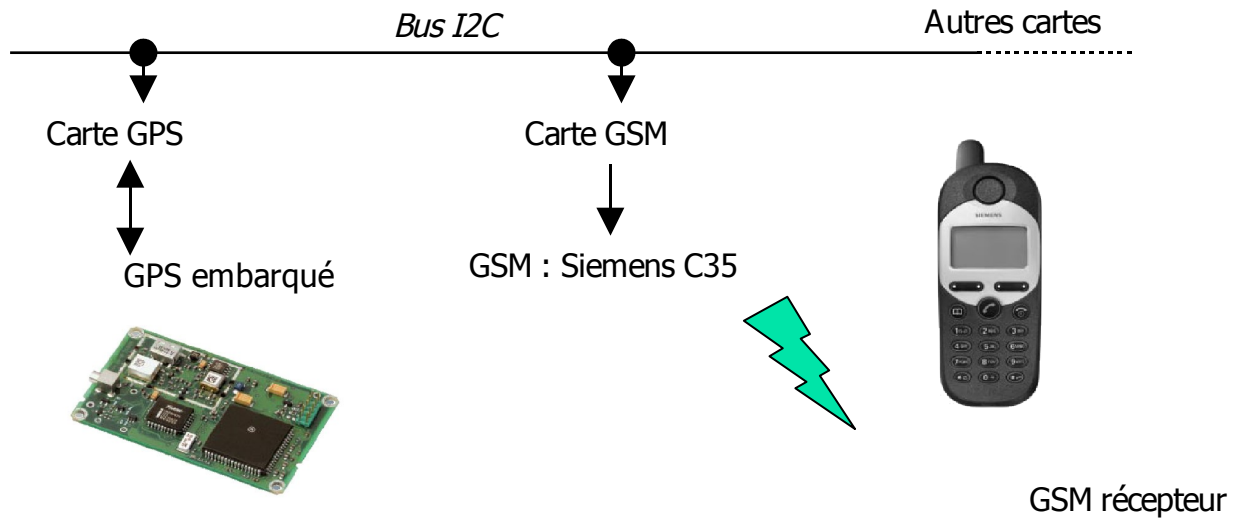
2.5.2.3. Alimentation

La batterie du mobile permet une veille d'au moins 24h et le vol est prévu pour durer 3h. Mais le mobile risque d'user inutilement la batterie en essayant de communiquer avec la station de base au cours du vol (un burst à pleine puissance au moins toutes les 6 min d'après la norme GSM). De plus le mobile qu'on nous prêterait sera probablement assez vieux et donc les performances de la batterie seront très dégradées.

Il est donc souhaitable d'avoir un moyen d'allumer et d'éteindre le portable.

On nous a conseillé d'allumer le mobile à des intervalles réguliers pour maintenir la batterie à sa température de fonctionnement (risque de non-rallumage si elle est trop froide).

2.5.2.4. Récupération des données GPS



L'architecture du ballon est centrée autour d'un bus I2C qui permet les communications entre les différentes expériences.

Une carte à base de microcontrôleur PIC 16F876 est consacrée à la gestion du module GPS et aux télémessures. Elle possède également une liaison I2C pour accéder aux convertisseurs A/N des diverses expériences. On utilisera cette liaison I2C pour transmettre les informations de position du message SMS.

La carte GPS est un périphérique I2C maître, la carte GSM est donc utilisée en périphérique esclave. Les adresses I2C 0x04 → 0x0A sont dédiées à des utilisations spéciales, on choisit 0x04 pour ce périphérique.

Les données venant du GPS seront transmises en ASCII, ce qui simplifie beaucoup leur utilisation dans un message texte SMS. De plus, en utilisant les trames NMEA du GPS, on travaille entièrement en mode texte, et on se passe donc de conversion flottant → texte très gourmandes en mémoire.

On transmettra donc, en ASCII :

Index			
0	Latitude	Degrés	10
1			1
2		Minutes	10
3			1
4			1/10
5			1/100
6			1/1000
7			1/10000
8	N/S (S ???)	'N'/S'	

9	Longitude	Degrés	100
10			10
11			1
12		Minutes	10
13			1
14			1/10
15			1/100
16			1/1000
17		1/10000	
18		W/E	'W'/E'
19	Altitude	Mètres	10000
20			1000
21			100
22			10
23			1
24	Heure	Heures	10
25			1
26		Minutes	10
27			1
28		Secondes	10
29			1
30	Validité	0 position non valide 1 position 2D valide 2 position 3D valide	

2.5.2.5. Envoi du SMS

- Interface GSM - PIC

L'envoi d'un SMS nécessite une liaison avec le mobile GSM. L'interface la plus utilisée sur les anciens mobiles est une liaison série asynchrone. La documentation concernant les brochages des téléphones et les paramètres de connexion est très difficile à trouver, les fabricants ne communiquent guère sur ces sujets.

Les commandes pour contrôler le mobile sont héritées des modems Hayes des PCs : ce sont les commandes AT. La documentation des commandes AT supportées par les mobiles est très variable selon les fabricants. Elles sont globalement standardisées, mais peuvent différer pour les fonctions spéciales.

- Composition et envoi du SMS

La commande AT permettant l'envoi d'un SMS est la suivante, sur toutes les documentations rencontrées

AT+CMGS

AT+CMGS	Send an SMS
Test command AT+CMGS=?	Response OK
Write command If PDU mode (+CMGF=0) +CMGS=<length><CR>PDU is given <ctrl-Z/ESC>	Parameter <length> Length of PDU <pdu> See "AT+CMGL" <mr> Message reference Response If sending is successful: +CMGS: <mr> If sending is not successful: +CMS ERROR: <err>

Siemens – AT Command Set Reference Manual

Pour envoyer les SMS, plusieurs modes sont à disposition. Entre autres le mode PDU – Protocol Data Unit – est le plus général et le mieux supporté. En contrepartie il est également plus difficile de l'utiliser, le format des messages PDU étant assez contraignant (alphabet 7-bits, hexa inversé). Ce mode fait l'objet de la recherche documentaire.

2.5.2.6. Mode PDU

- Champs du message

Champ	Description
TP-MTI	First octet
TP-MR	TP-Message-Reference
TP-DA	TP-Destination-Address
TP-PID	TP-Protocol-Identifier
TP-DCS	TP-Data-Coding-Scheme
TP-VP	TP-Validity-Period
TP-UDL	TP-User-Data-Length
TP-UD	TP-User-Data

On se restreint à l'étude des champs utilisés par la commande AT. Il existe d'autres champs (plus proche du réseau) que le téléphone GSM se charge de remplir.

Premier octet : TP-MTI

TP-MTI	TP-Message-Type-Indicator	2b	Parameter describing the message type.
TP-RD	TP-Reject-Duplicates	b	Parameter indicating whether or not the SC shall accept an SMS-SUBMIT for an SM still held in the SC which has the same TP-MR and the same TP-DA as a previously submitted SM from the same OA
TP-VPF	TP-Validity-Period-Format	2b	Parameter indicating whether or not the TP-VP field is present.
TP-RP	TP-Reply-Path	b	Parameter indicating the request for Reply Path.
TP-UDHI	TP-User-Data-Header-Indicator	B	Parameter indicating that the TP-UD field contains a Header.
TP-SRR	TP-Status-Report-Request	b	Parameter indicating if the MS is requesting a status report.

bit1	bit0	Message type
0	0	SMS-DELIVER (in the direction SC to MS)
0	0	SMS-DELIVER REPORT (in the direction MS to SC)
1	0	SMS-STATUS-REPORT (in the direction SC to MS)
1	0	SMS-COMMAND (in the direction MS to SC)
0	1	SMS-SUBMIT (in the direction MS to SC)
0	1	SMS-SUBMIT-REPORT (in the direction SC to MS)
1	1	Reserved

MTI = 0x01

VP Field Validity Period Format

bit4	bit3	
0	0	TP-VP field not present
1	0	TP-VP field present - relative format

0	1	TP-VP field present - enhanced format
1	1	TP-VP field present - absolute format

On utilise le mode Relative Format : 0x10

RP = 0x10

UDHI = 0

SRR = 0

Donc, au final, le premier octet devra avoir un code : 0b00010001 = 0x11

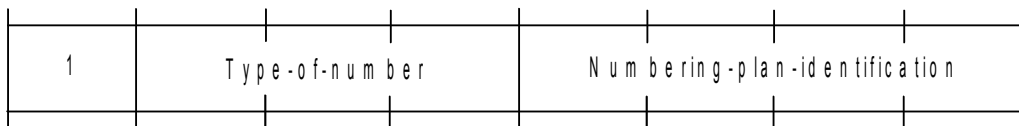
Message Reference TP-MR

C'est un identifiant différent pour chaque SMS envoyé, il est automatiquement modifié par le mobile. On peut donc le mettre à n'importe quelle valeur.

Destination Address TP-DA

C'est le numéro de téléphone du destinataire. Les numéros de téléphone sont codés un format propre à la norme GSM

Premier octet : type de numéro de téléphone



Type-of-number:

Bits 6 5 4

0 0 0	Unknown ¹⁾
0 0 1	International number ²⁾
0 1 0	National number ³⁾
0 1 1	Network specific number ⁴⁾
1 0 0	Subscriber number ⁵⁾
1 0 1	Alphanumeric, (coded according to GSM TS 03.38 7-bit default alphabet)
1 1 0	Abbreviated number
1 1 1	Reserved for extension

Numbering-plan-identification (applies for Type-of-number = 000,001,010)

Bits 3 2 1 0

0 0 0 0	Unknown
0 0 0 1	ISDN/telephone numbering plan (E.164/E.163)
0 0 1 1	Data numbering plan (X.121)
0 1 0 0	Telex numbering plan
1 0 0 0	National numbering plan
1 0 0 1	Private numbering plan
1 0 1 0	ERMES numbering plan (ETSI DE/PS 3 01-3)
1 1 1 1	Reserved for extension

All other values are reserved.

Pour notre projet, nous utiliserons un numéro de téléphone international (au cas où le ballon tombe à l'étranger). On configurera donc ce champ à la valeur

0b10010001 = 0x91

Deuxième octet : longueur

Longueur du numéro de téléphone.

bits	TP-Protocol-Identifier (TP-PID) usage
7 6	
0 0	Assigns bits 0..5 as defined below
0 1	Assigns bits 0..5 as defined below
1 0	reserved
1 1	Assigns bits 0-5 for SC specific use

In the case where bit 7 = 0, bit 6 = 1, bits 5..0 are used as defined below

5..0	
000000	Short Message Type 0
000001	Replace Short Message Type 1
000010	Replace Short Message Type 2
000011	Replace Short Message Type 3
000100	Replace Short Message Type 4
000101	Replace Short Message Type 5
000110	Replace Short Message Type 6
000111	Replace Short Message Type 7
001000..011110	Reserved
011111	Return Call Message
100000..111100	Reserved
111101	ME Data download
111110	ME De-personalization Short Message
111111	SIM Data download

Nous envoyons un SMS standard (les autres types étant utilisés sur les portables de nouvelles génération), de type 0. Donc ce champ doit être à la valeur : 0b00000000 = 0x00

TP Validity Period TP-VP

C'est la durée de stockage du SMS dans le centre serveur, tant qu'il n'a pas été envoyé.

TP-VP value	Validity period value
0 to 143	(TP-VP + 1) x 5 minutes (i.e. 5 minutes intervals up to 12 hours)
144 to 167	12 hours + ((TP-VP - 143) x 30 minutes)
168 to 196	(TP-VP - 166) x 1 day
197 to 255	(TP-VP - 192) x 1 week

255 = 58 semaines.

Codage du texte : alphabet 7-bits

Texte original, en ASCII

A7	A6	A5	A4	A3	A2	A1	A0	B7	B6	B5	B4	B3	B2	B1	B0		I7	I6	I5	I4	I3	I2	I1	I0
Caractère 0								Caractère 1								Caractère 7								

En codage 7 bits

B0	A6	A5	A4	A3	A2	A1	A0	C1	C0	B5	B4	B3	B2	B1	B0		I6	I5	I4	I3	I2	I1	I0	H6
Caractère 0								Caractère 1								Caractère 6								

2.5.2.7. Création d'un programme pour tester le mode PDU

Pour mieux prendre en main la composition en mode PDU, nous avons écrit un programme Java permettant la communication avec le mobile au moyen du port infrarouge. Ce programme permet d'envoyer des commandes AT et donc un message SMS. Il est beaucoup plus souple à utiliser qu'un codage directement sur le PIC, avec à chaque fois les longues étapes de compilation / programmation.

Il s'est montré indispensable pour comprendre les champs des messages PDU et le codage 7-bits de l'alphabet.

Le code source est donné en annexe.

2.5.2.8. Contrôle de l'alimentation

Certains des mobiles que nous avons utilisés s'allument dès application d'une tension sur leur connecteur d'interface. En revanche, le modèle retenu reste éteint lors de l'envoi des commandes AT ou lors d'une alimentation extérieure.

Comme on souhaite contrôler l'allumage/extinction du portable, la seule solution restante est de court-circuiter le bouton poussoir du clavier utilisé pour l'allumage. N'étant pas encore sûr du moyen à utiliser, nous plaçons deux relais sur la carte PIC qui seront ensuite utilisés selon le modèle du portable retenu.

2.5.2.9. Microcontrôleur

On réalise un montage standard autour d'un PIC 16F876. Comme les opérations à effectuer ne sont pas très complexes, on peut choisir n'importe quelle fréquence de fonctionnement. Pour limiter la consommation, on prend 10,5984MHz (le premier qu'on ait trouvé dans le tiroir).

On ajoute un convertisseur de niveau MAX233 pour interfacier le PIC avec un port série RS232. L'utilisation d'un terminal sur PC s'avère très pratique pour mettre au point le logiciel, surtout que le compilateur gère la fonction 'printf(...)' vers le port série. Deux jumpers permettent de diriger les communications selon 3 modes de fonctionnement :

- PIC – GSM : normal
- PIC – RS232 : essais logiciels
- GSM – RS232 : essais mobile

Une led dite « led vie » sert de témoin de bon fonctionnement. Selon son état (fixe, clignotante) on peut connaître dans quelle phase est le programme.

2.5.2.10. Logiciel embarqué

La programmation est réalisée en C.

Remarques : le compilateur optimise parfois à tort, et on a notamment rencontré ce problème :

```
{  
...  
    i2c_read() ; // Attente d'un octet sur le bus I2C  
}
```

qui aurait du se traduire par un test du flag de réception d'un octet, est en fait compilé en nop. Pour obtenir la bonne compilation, il faut utiliser une affectation, le compilateur comprend alors que l'instruction est importante.

```
{  
    ...  
    t = i2c_read() ;  
}
```

fonctionne bien.

Par la suite, on s'est passé au maximum des fonctions du compilateur pour faire des accès directs aux registres du PIC.

Pour nos essais, nous n'avons pas utilisé le critère 'être dans le dernier kilomètre' pour envoyer des SMS. Cela pourra se faire avec un simulateur de trame GPS, en cours de développement.

2.5.3. Tests

2.5.3.1. Tests unitaires

Nous avons réalisé des petits programmes qui testent le bon fonctionnement de chaque périphérique du pic (led vie, relais, port série, bus I2C).

2.5.3.2. Communications GSM – PIC

Espion sur port série :

```
AT+CMGF=0  
OK  
AT+CSMS=0  
+CSMS: 1,1,1  
  
OK  
AT+CMGS=63  
>  
07913386094000F011C40B913386905060F60000FF37CC305D07A2E19C34998B76C3D14  
00DE6DB  
AD03C160B2222CE7BAD56C375023C8A6836030180CD6063562331DACA69BD500  
  
+CMGS: 187  
OK
```

2.5.4. Conclusion

Nous pouvons envoyer avec succès des SMS contenant la position GPS du ballon. Il reste encore quelques modifications à faire sur le logiciel pour prendre en compte les conditions d'envoi du SMS (dernier kilomètre, en descente). Nous devons aussi régler le problème de l'alimentation du mobile, mais nous préférons le faire après la présentation, par crainte d'endommager le mobile (il faut en effet souder au niveau du bouton marche/arrêt, sur des pistes assez fragiles).

3. Alimentations

3.1. Présentation

L'alimentation d'un ballon est la partie essentielle pour le succès d'un projet ballon. Elle est loin d'être évidente et peut mener à l'échec d'un projet. Dans un ballon, elle est soumise à plusieurs contraintes, que l'on a pu rencontrées :

- La masse limitée du ballon, qui oblige à limiter le nombre de piles et à surveiller le rendement
- La CEM, alimenter de l'électronique d'instrumentation et un émetteur à partir de la même source n'est pas évident.
- Les conditions atmosphériques, les caractéristiques des piles n'étant déjà pas trop documentées à température ambiante, elles sont inconnues aux basses températures.

3.2. Bilan électrique

3.2.1. Besoins

Carte	Tensions	Consommation	Capacité requise, vol de 3,5h
PIC1	4,5 à 5,5V	50mA	1,1 A.h
GPS	4,75 à 5,2V Ripple 50mV c/c	250mA	
Modulateur FSK	12V	20mA	
Environnement	+8V -8V	50mA	200 mA.h
Geiger	9V	50mA	
Emetteur	9 à 12V	250mA	1 A.h

3.2.2. Ressources

Autant de piles 4,5 que l'on veut. C'est une limite que l'on s'impose, justifiée par l'expérience de la tenue des ces piles au froid (en particulier des Duracell, seules piles du commerce à être presque correctement documentées). Mais c'est vrai qu'on sera limité sur les tensions disponibles en entrée (4,5V, 9V, 13,5V).

3.3. Synthèse

La carte PIC (avec le module GPS) est la plus gourmande en énergie. C'est également celle sur laquelle se trouve toute l'électronique. On choisira donc une alimentation propre et puissante.

Les cartes environnement sont essentiellement en analogique. Elles sont aussi très sensible au bruit, mais consomment peu. Par contre elles nécessitent une alimentation symétrique. On choisira une alimentation symétrique propre.

L'émetteur est un gros consommateur, mais possède son propre régulateur. Il faudra l'essayer avec une alimentation un peu 'sale'.

Devant l'étendue de toutes les tensions demandées, la carte alimentation fournira les tensions suivantes, les autres étant générées à partir de celles-ci au niveau local.

GND, +12V, -12V, +5V

3.4. Conception

3.4.1. Contraintes

L'utilisation d'alimentations à découpage, outre l'aspect formateur (jamais réalisées auparavant), offre un excellent rendement et donc une optimisation des piles. On l'utilisera pour les systèmes les plus consommateurs d'énergie. Il s'agit donc du PIC et du GPS.

Toutefois le bruit qu'elles génèrent peut dégrader les performances des cartes alimentées. Pour les cartes les plus sensibles on placera un régulateur linéaire LDO en sortie de l'alimentation à découpage. Ces cartes seront donc pourvues localement d'un LDO qui fournira également les tensions demandées par les circuits.

On essaiera d'utiliser au maximum des combinaisons de piles en série, qui évitent l'autodécharge d'une pile dans une autre.

3.4.2. Régulation à découpage

Alimentation processeur/GPS +5V

400mA, 5V :

fréquence de découpage : 200 kHz.

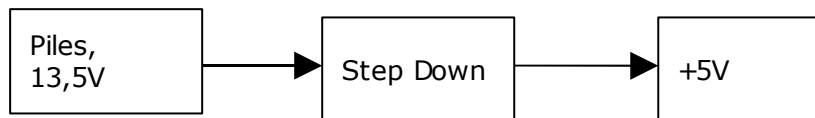
Inductance propre : 100µh

Energie stockée : $\frac{1}{2}LI^2$

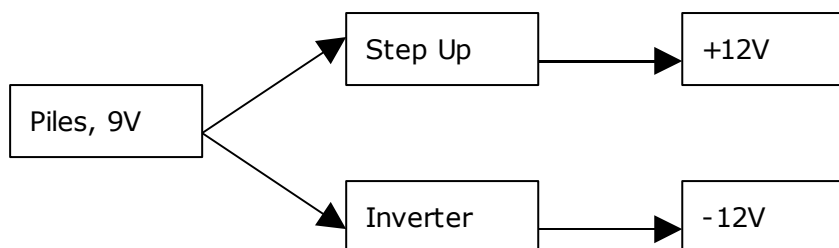
Régulation: 5V, rendement 85%, $I_{max}=500mA$

→ 2,9W.

Pour un cycle, $E_0 = 2,9/200.000 = \frac{1}{2} LI^2 \rightarrow I=550mA$



Alimentation +12V/-12V



En utilisant une source de 9V, on utilisera uniquement des convertisseurs step-up. En utilisant une source de 13,5V et convertisseur Step-Down, on peut imaginer que la température et l'usure des piles puissent faire baisser la tension en dessous de 12V, et donc le mode step-down du convertisseur ne pourrait plus fonctionner.

Une configuration Sepic de l'alimentation à découpage aurait permis un fonctionnement dans les deux cas, mais elle est plus complexe et pas vraiment justifiée.

3.5. Réalisation

3.5.1. Prototype

Pour valider la possibilité d'alimenter les circuits 'sensibles' par une alimentation à découpage, on réalise un petit circuit d'alimentation 5V.

Ne pas oublier de bien dimensionner l'inductance ! Le premier essai, où je demandais ambitieusement 1A dans une résistance de puissance, s'est terminé par une fumée malodorante sortant de l'inductance. En regardant de plus près la datasheet, celle-ci ne tolérait qu'une dissipation associée à un courant de 600mA...

Si l'alimentation donnait satisfaction pour la partie logique (aucun problème avec le GPS), ce fut en revanche une catastrophe sur la partie analogique, lorsque l'alimentation marchait (pas à tout les coups).

3.5.2. Conclusion

En définitive, la réalisation d'alimentations à découpage s'est soldée par un échec, le bruit en sortie étant trop important pour l'électronique analogique, et la réalisation d'un bon filtrage demandait trop de temps et de moyens.

Nous nous sommes donc tournés vers des régulateurs à découpage tout intégrés, de chez Texas Instruments, auxquels on a ajouté un filtre LC en PI, avec des ferrites sur les lignes d'alimentation.

3.6. Bilan de puissance

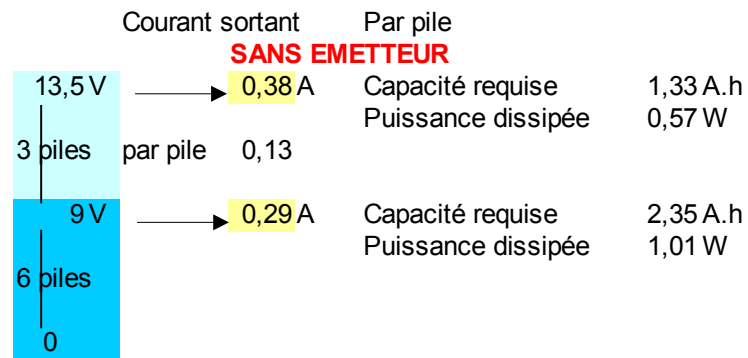
3.6.1. Estimé

	Batteries			Rendement conversion	Mesure Alimentation		
	V (V)	I (A)	P (W)		V	I	P
Kiwi	13,50	0,210	2,84				
5	13,50	0,148	2,00	0,75	5,00	0,300	1,50
-12	9,00	0,000	0,00	0,85	12,00	0,000	0,00
12	9,00	0,111	1,00	0,60	12,00	0,050	0,60

Tension	Intensité(A)	Puissance (W)	Par pile de 1,5V (W)	Capacité requise (A.h) par pile	Capacité annoncée MN1203 500mA
9,00	0,11	1,00	0,17	0,39	1,5A.h
13,50	0,36	4,84	0,54	1,25	1,5A.h
Durée du vol	3,50				

3.6.2. Mesuré

Durée de vol 3,5 h



Le bloc de 6 piles 9V est donc doublé.

Pour l'émetteur, on a choisi 9 piles 1,5V comme source exclusive.

4. Téléméasures

4.1. Emission

L'émission se fait par modulation FSK 600 bauds sur l'entrée externe de l'émetteur KIWI.

4.2. Modulation FSK

Les trames sont transmises par modulation FSK avec 2 fréquences clefs à $f_0=900\text{Hz}$ et $f_1=1,5\text{kHz}$, autorisant un débit maximum théorique de 600 bauds.

Le circuit sera basé sur un VCO (Voltage Controlled Oscillator) à XR2206, montage courant dans les clubs spatiaux.

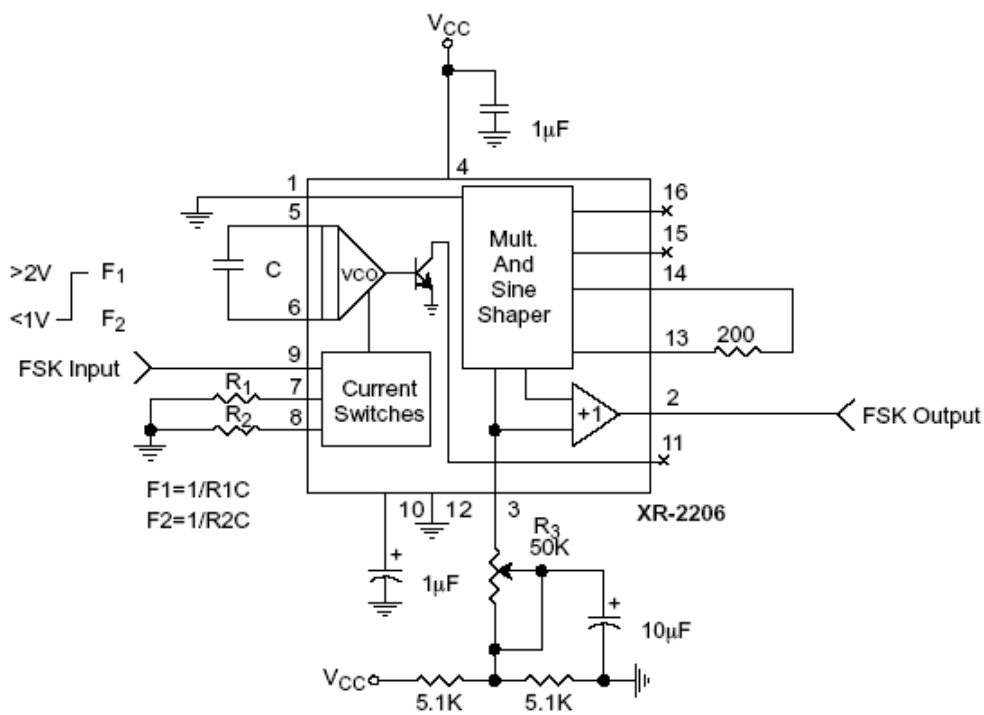


Figure 13. Sinusoidal FSK Generator

- $f_0 = 1 / R1.C$
- $f_1 = 1 / R2.C$

La datasheet recommande :

- C [1nF, 100μF]
- R [1kOhm, 2Mohm]

On fixe C = 100nF, et on trouve

R1 = 11,111kOhm et R2 = 6,667kOhm

4.3. Trames télémessures

4.3.1. Spécifications

Emission trame GPS : toutes les 5 secondes

Emission température, pression, radioactivité : toutes les secondes

Emission statut général (à définir, reboot, tensions piles, ...) : toutes les 10 secondes.

Débit 600 bauds pour une meilleure densité de puissance spectrale

4.3.2. Protocole

Format de transmission bas niveau

- 600 bauds, format RS232
- 1 bit de start
- 8 bits de données
- 1 bit de stop
- sur modulation FSK 600 bauds 9000/15000.
- Durée de transmission (1 octet de donnée) : 16,7ms

Tolérance sur le baudrate

$$\frac{9}{Baud \pm \Delta Baud} = \frac{9}{Baud} + \frac{1}{2 \cdot Baud}$$
$$\frac{Baud}{Baud \pm \Delta Baud} = \frac{19}{18}$$
$$1 \pm \frac{\Delta Baud}{Baud} = \frac{18}{19}$$
$$\frac{\Delta Baud}{Baud} = \pm \left(\frac{18}{19} - 1 \right) = \pm 5,3\%$$

En négligeant les temps de montées / descentes (qq centaines de nanosecondes), on considère la synchronisation de l'UART de réception sur le front descendant du START.

baudMin = 569

baudMax = 631

4.3.3. Format d'encapsulation des trames

- Octets de synchronisation : 0xFF 0xFF 0xAA
- Type de trame (1 octet)
- Données
- Checksum (XOR des données)

Types de données

[xxx] : x caractère ascii

entier signé / non signé : entier binaire MSB first

4.3.3.1. Trame GPS NMEA variante 'Light'

Trame NMEA non compatible (quoique...), mais beaucoup plus légère.

Avantages : on garde la représentation 'caractère', lisible sous un terminal et directement exploitable par le GSM, tout en diminuant la durée totale d'une trame.

On garde le séparateur de chaînes ','.

Offset	Format	Taille (octets)	Description
0	0xFF,0xFF,0xAA	3	Synchronisation
	0x91	1	Type de trame (GPS)
	[hhmmss],'	7	Heure
	[ddmm].'[mmmm],'	10	Latitude
	['N'/S],'	2	Hémisphère Nord/Sud (bof...)
	[dddmm].'[mmmm],'	11	Longitude
	['E'/W],'	2	Secteur Est/Ouest
	['0'/1],'	2	Position valide
	[nn],'	3	Nb de satellites utilisés
	[nn],'	3	HDOP (partie entière uniquement)
	[aaaa],'	6	Altitude en mètres
	Entier non signé	1	Checksum (XOR)
		51	

Durée : 852 ms (quand même...)

4.3.3.2. Trame expériences

Type de trame : 0x13

Offset	Type	Taille (octets)	Description
0	0xFF,0xFF,0xAA	3	Synchronisation
3	0x13	1	Type de trame (Expériences)
4	Entier signé	1	Température intérieure en °C
6	Entier signé	1	Température extérieure en °C
7	Entier non signé	2	Pression en Pa
9	Entier non signé	2	Nombre de coups Geiger entre deux envois de trame
	Entier non signé	1	Checksum

Durée de transmission : 184 ms

4.3.3.3. Trame statut général

Type de trame : 0xB4

Offset	Type	Taille (octets)	Description
0	0xFF,0xFF,0xAA	3	Synchronisation
3	0xB4	1	Type de trame (Statut)
4	Entier non signé	1	Tension 1
5	Entier non signé	1	Tension 2

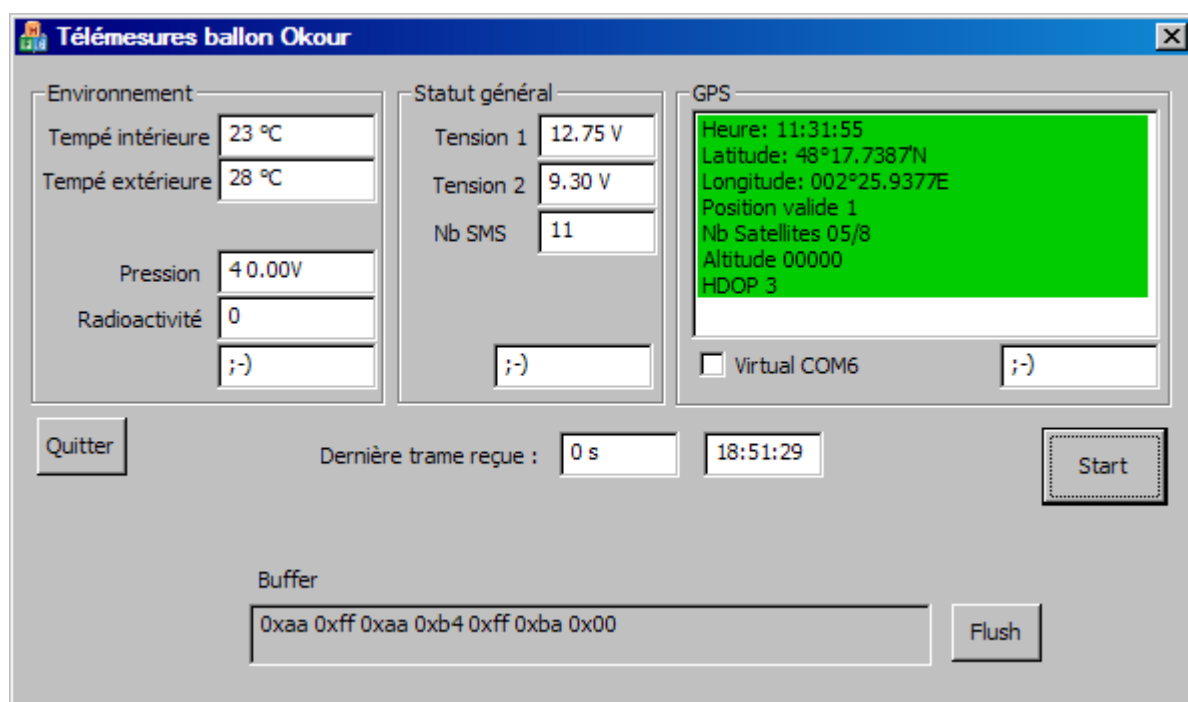
6	Entier non signé	1	Tension 3
7	Entier non signé	1	Nb d'envois de SMS
8		1	Checksum

4.4. Réception

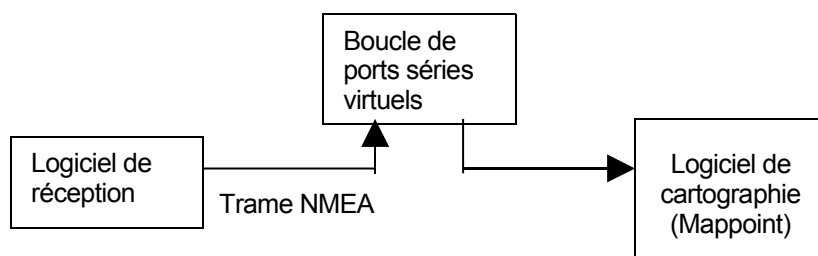
4.4.1. Logiciel de réception

Le logiciel de réception est une application écrite sous Visual C++. Je tiens à remercier Jérôme Hamm pour son code du projet Axelle (CLES-FACIL).

Les données de chaque expérience sont affichées numériquement.



La position GPS est reportée sur un logiciel de cartographie quelconque en utilisant un port série virtuel. Le logiciel de réception reçoit les trames GPS, les encode sous une forme NMEA standard puis les réémet sur un port série virtuel. Un logiciel en shareware boucle ce port série vers un autre port virtuel sur lequel le logiciel de cartographie recevra les trames NMEA. Au départ la gestion du port série virtuel devait être gérée par le logiciel de réception, mais, par manque de temps, il a fallu recourir à un shareware.



5. Lâcher du ballon

5.1. Cadre du lancement

Une première date de lancement avait été envisagée le 31 juillet 2004 lors de la Rencontre Nationale des clubs Espace 2004 à Flixecourt. La phase de tests de télémétrie ayant fait apparaître des dysfonctionnements, le lancement a été reporté au 19 septembre 2004 depuis la base de loisirs de Buthiers. Le lâcher s'est effectué sur le parking et la réception fixe se faisait depuis l'observatoire. Nous disposions d'un système de réception mobile. Une voiture était équipée d'un récepteur AR8200, d'un démodulateur KIWI et du logiciel de réception dédié au ballon ; une seconde voiture était équipée d'un récepteur « Gonzague », d'un onduleur 12V - 220V et du logiciel Keasy.



5.2. Récupération du ballon

Les deux voitures étaient équipées d'un logiciel de cartographie qui permettait de visualiser la position GPS de la voiture et du ballon. L'une a emprunté la nationale, l'autre l'autoroute ; les deux sont arrivées à proximité du point de chute environ 10 minutes après la réception de la dernière trame. La poursuite du ballon a donc été un succès. Son état général est très bon, les cartes n'ont pas bougé, seule l'antenne filaire s'est déformée, entraînant la fin de l'émission des télémétries après l'impact.



Illustration 11: Nico l'a vu,, Manu l'a entendu

5.3. Récupération des résultats

Le logiciel KEasy a bien fonctionné tout au long du vol et a aidé à la récupération du ballon. Malheureusement, les données reçues n'étant pas enregistrées (ou étant effacées lors d'un nouveau lancement), le logiciel ne nous a été d'aucune utilité pour l'exploitation des résultats.

En raison d'un problème de réception avec le récepteur mobile de la deuxième voiture, la première phase de vol n'a pas été reçue mais l'enregistrement des données a pu être correct à partir de la fin de la montée et jusqu'à l'atterrissage, ce qui permis de localiser le ballon au sol avec précision.

Le système qui était considéré le plus fiable, à savoir la station de réception fixe, a rencontré de nombreux problèmes encore inexpliqués (connecteur d'antenne défaillant ?). L'erreur principale a été de partir suivre le ballon en voiture sans avoir regardé le bon fonctionnement de la station fixe (qui n'était qu'à 10 minutes du parking). Comme la personne s'occupant de la station fixe n'était pas membre du projet, elle n'a pas noté le comportement anormal de la réception. Il s'agit là de la plus grosse erreur commise sur ce projet.

6. Exploitation des résultats

6.1. *Traitement des données*

Toutes les données sont exportées vers une structure unique sous Matlab, qui permet une manipulation plus simple qu'Excel.

6.2. *Données générales sur le vol*

Altitude maximale : 21 738 m

Durée de vol : 1h58 min (ballon beaucoup gonflé)

Distance directe parcourue :

6.3. Atmosphère

6.3.1. Température

6.3.1.1. Température intérieure :

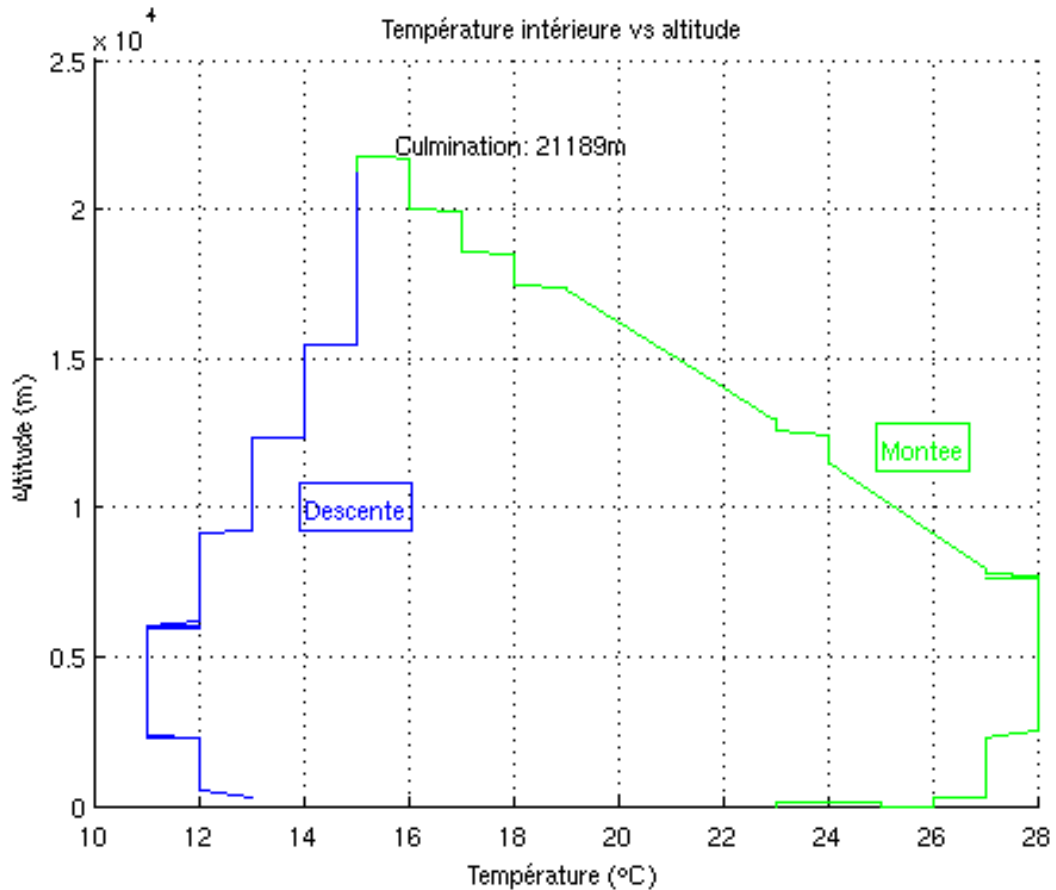


Illustration 12: Température intérieure

La nacelle s'est refroidie pendant toute la durée de la montée à partir de 8000m, puis s'est réchauffée en descente, en dessous de 6000m. La température de la nacelle est restée toujours très supérieure à celle de l'atmosphère durant quasiment tout le vol. Au-delà de 8000m, la température de l'atmosphère standard est inférieure à 12°C, température minimale atteinte par la nacelle. Il y a donc eu un refroidissement continu de la nacelle par conduction thermique. Les hausses de température observées dans les altitudes basses sont sûrement dues aux transferts de chaleur par rayonnement. La hausse due au rayonnement apparaît plus tardivement lors de la descente, car la vitesse de descente est plus importante que celle de montée à ces altitudes.

L'amplitude du refroidissement a été plus importante durant la montée, car la vitesse ascensionnelle y est moins importante, donc les échanges thermiques durent plus longtemps.

6.3.1.2. Température extérieure

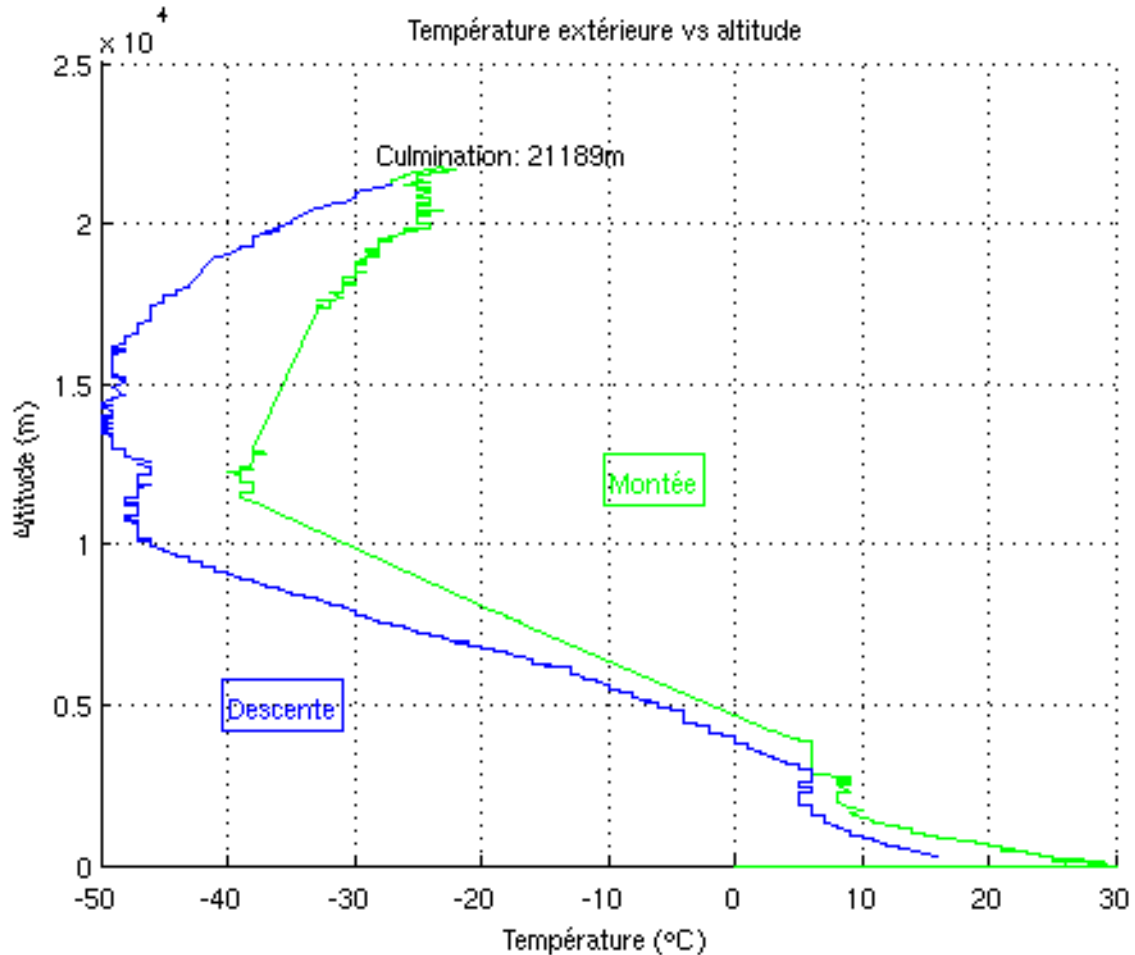


Illustration 13: Température extérieure

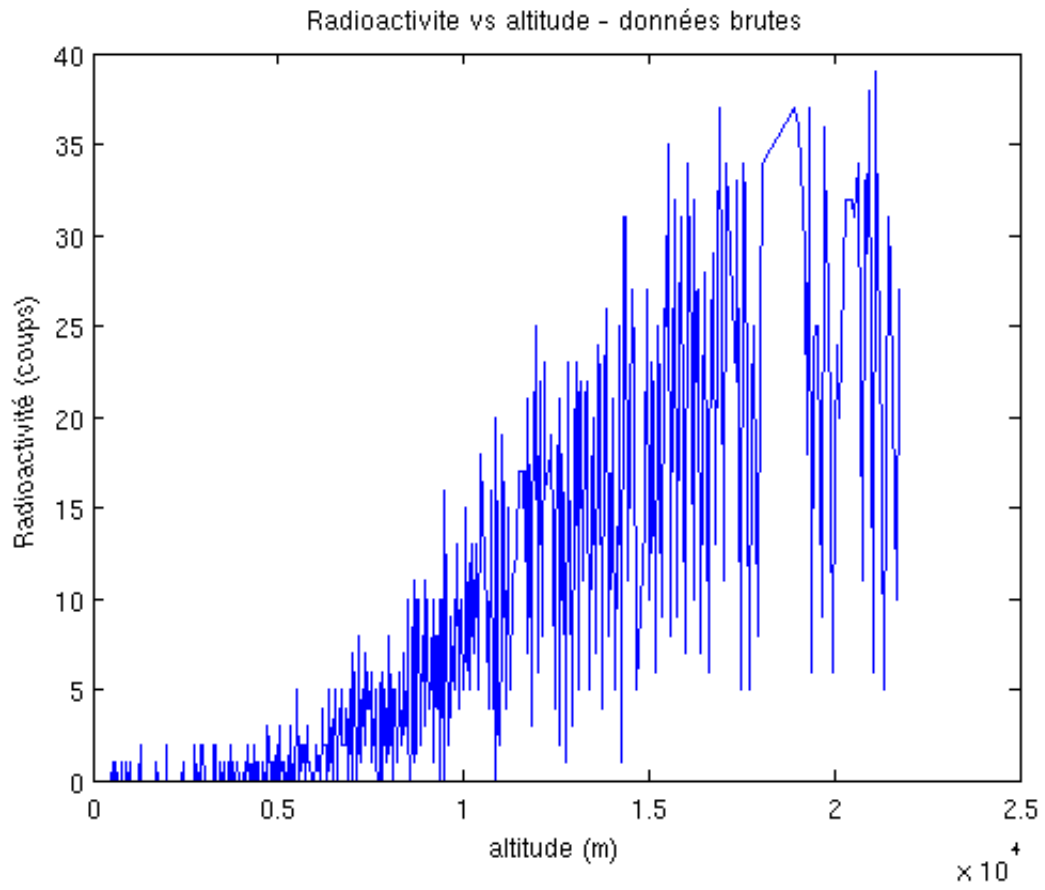
La température extérieure est conforme au modèle standard, même si la température minimale reste assez élevée. La différence de température entre la montée et la descente, à même altitude, s'explique par l'inertie thermique des capteurs.

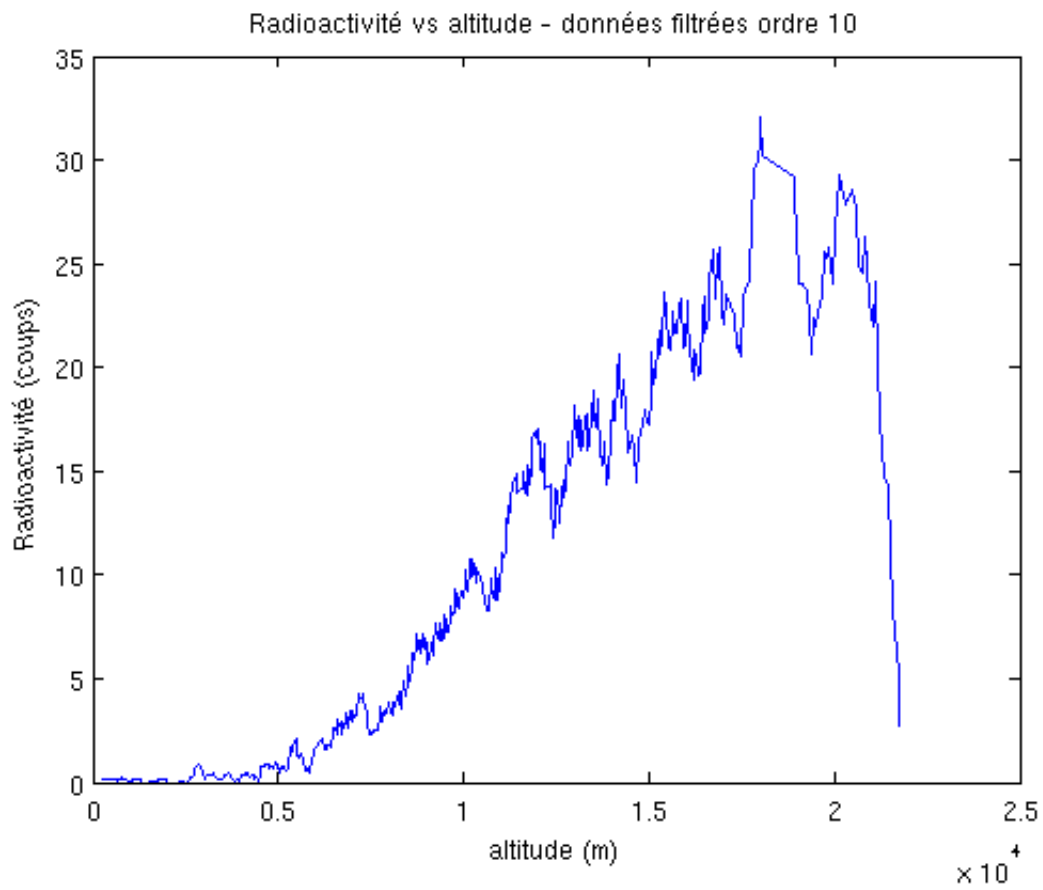
6.3.2. Radioactivité

Les données transmises sur la radioactivité sont toutes entachées d'une erreur !!

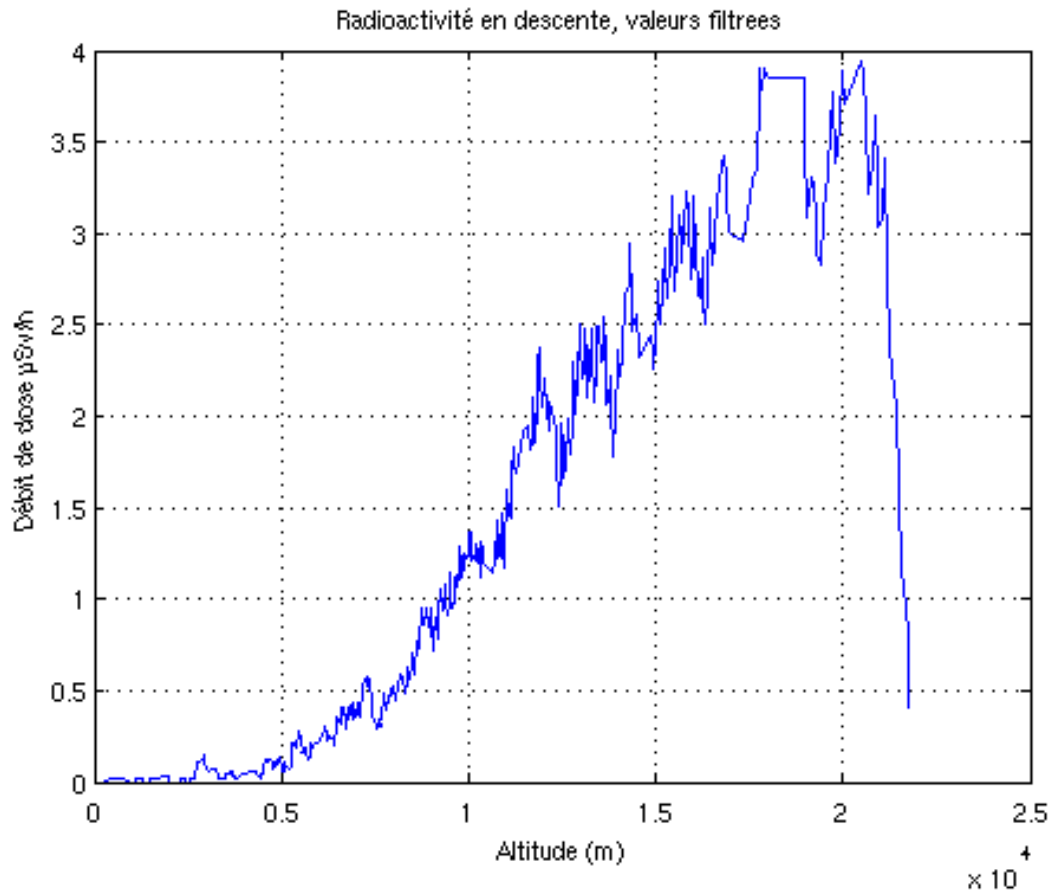
En effet, le programme compte le nombre d'impulsions reçues entre 2 envois de trame. Les deux envois devaient se faire à 3 secondes d'intervalles, ce qui permettait de retrouver le nombre de coups par seconde. Cependant, peut être à cause de problèmes d'interruptions sans priorité, cette période est beaucoup plus aléatoire, de 3 à 5 secondes... Par suite, il est impossible de remonter au début de dose de manière fiable.

On donne tout de même les graphes des valeurs reçues, en nombre de coups entre deux envois, sans connaître précisément la période d'envoi (3 secondes à 80%)





On corrige ces erreurs en rejetant systématiquement toutes les périodes d'acquisitions différentes de 3 et 5 secondes, à l'aide d'un petit script Matlab. Une centaine de mesures sont ainsi éliminées. Les mesures restantes sont exactes.



Valeurs corrigées : Les dernières mesures (chute brutale) ne sont pas en prendre en compte et sont le résultat du filtrage.

On observe un maximum, probablement le pic découvert par Pfozter en 1936, aux alentours de 15-20 km.

Avec la relation théorique $1 \text{ cps} = 0,46 \mu\text{Sv/h}$, le débit de dose maximum est de $3,9 \mu\text{Sv/h}$, autour de 17 km d'altitude.

6.4. Trajectoires

6.4.1. Vitesse ascensionnelle

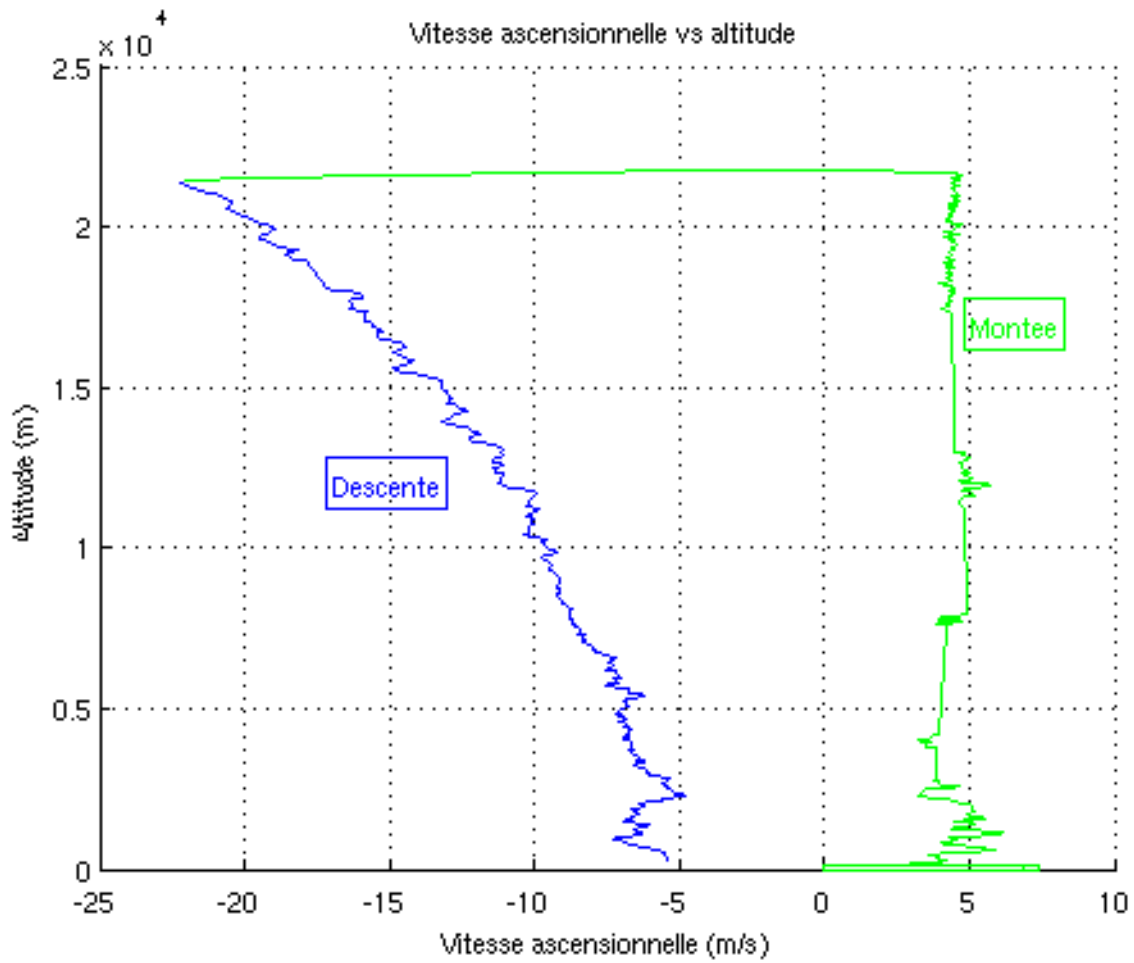


Illustration 14: Vitesse ascensionnelle en altitude

On remarque une vitesse de montée bien constante, autour de 5 m/s, et une vitesse de descente qui suit bien celle de la masse volumique de l'air.

6.4.2. Trajectoire sol

Trajectoire du ballon, ascension en rouge et descente en orange

Trajectoire de la voiture, en vert. Le petit crochet à Nemours, sur un parking de McDo, a permis de configurer convenablement le récepteur pour recevoir toutes les télémessures. Le ballon était alors au dessus de Chéroy.



7. Bilan

7.1. Organisation du projet

7.1.1. Problèmes techniques

Les plus gros problèmes techniques sont venus de la réception des télémesures. En particulier, l'utilisation de 3 trames différentes, envoyées avec des périodes différentes et surtout *sans transmettre une référence de temps (heure, numéro de trame)* fut une erreur énorme, à la limite du compréhensible !

Un autre point qui a pu sembler délicat est celui des alimentations, peut être dimensionnées un peu juste.

L'absence totale d'envois de SMS par le ballon le jour du vol reste un mystère. Ils sont peut être à relier avec la carte SIM du téléphone, utilisée maintenant par Nicolas et qui présente des problèmes répétés (impossible de trouver la borne la plus proche, redirection immédiate vers le répondeur même en cas de bonne couverture, etc...)

7.1.2. Carte environnement

L'abandon de la carte environnement, à quelques jours du lâcher et suite à des problèmes d'interférences radiofréquences (probablement des couplages entre les champs émis par antenne, et les boucles créées par les capteurs, avec des impédances communes), ou par retour de HF sur les alimentations reste un point décevant du ballon, surtout vu le soin particulier apporté à cette carte, de sa conception (y compris CEM !) jusqu'à son étalonnage. Les couplages par impédance commune seront à surveiller de plus près la prochaine fois, et l'utilisation de liaisons différentielles sera à privilégier.

7.2. Résultats

7.2.1. International Astronautical Congress

A la suite d'un appel à présentations de l'ESA pour le Congrès International d'Astronautique, le ballon OKOUR a été retenu, et Claire a été choisie pour présenter le projet à Vancouver (Canada).

7.2.2. Télémesures

Globalement, et après de très longues heures pour dater et identifier les trames reçues par la seule des 3 stations ayant enregistré les données, les résultats des télémesures sont assez satisfaisants.

7.3. Conclusion

Le ballon a occupé une place importante pour nous tous et durant toute une année, en détachement quasi-compét des études. L'organisation de projet, les connaissances théoriques acquises, les difficultés levées et le savoir-faire développé marquent déjà le projet d'une très grande réussite.

Après de nombreuses péripéties, le ballon a pu prendre son envol. Malgré quelques difficultés sur la fin, lors de la réception et l'analyse des télémesures, les résultats obtenus sont intéressants et en tout cas confirment le succès technique du projet.

La sélection du projet Okour par l'ESA et sa présentation à Vancouver, lors de l'IAC 2004, ont été la consécration finale du projet.

8. Remerciements

Nos remerciements vont en premier au CNES et à Planète Sciences, pour l'occasion unique qu'ils offrent aux jeunes. Nous remercions également l'ESA et l'IAF qui nous ont permis de présenter le ballon à l'International Astronautical Congress de Vancouver.

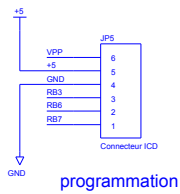
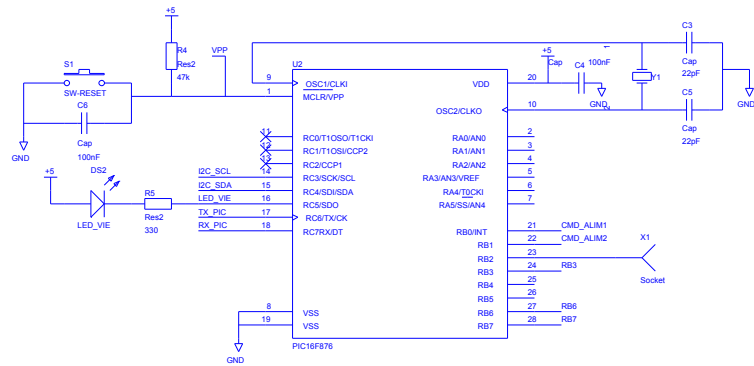
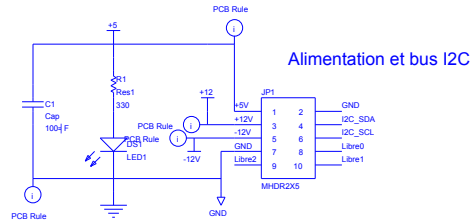
Plus particulièrement, nous souhaitons remercier chaleureusement:

- Les experts divers en électronique, informatique et autres :
Gérard Auvray, Nicolas Chaléroux, Jérôme Hamm, Manu Jolly, Francis Lésel, Etienne Maier, Michel Maignan
- Aux permanents et amis du mercredi soir
Etienne, Vincent, Laurent et les autres
- Aux profs de Supélec qui nous ont soutenus :
M. Benabes et M. Gauthier
- Aux participants de la chasse au ballon et du lâcher
Manu, Nico, Caroline, La famille Costy, La famille Guérad

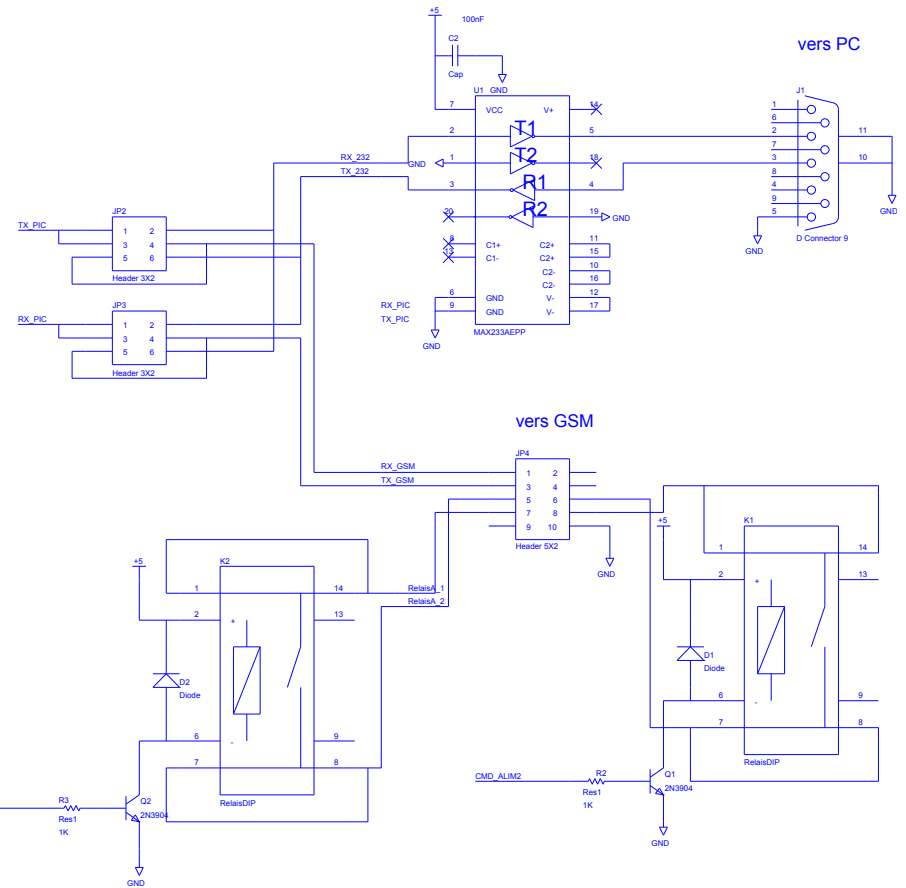
9. Annexes

9.1. *Expérience GSM*

9.1.1. Schémas



Convertisseur de niveaux



9.1.2. Code source

9.1.2.1. sms.c

```
/**
 *
 * Fonctions pour l'envoi du SMS
 *
 */
#ifdef __SMS_C__
#define __SMS_C__

#include "ledvie.c"

#define START_PDU "07913386094000F011000B913336920570F80000FF"

/**
 * conversion Alphabet 7-bits
 *
 * ATTENTION: str doit être alloué pour contenir suffisamment de
 * caractères
 * pour être complété jusqu'au multiple de 8 supérieur.
 */
void
conversionAlphabet (char *str, int8 len, int8 * len7bits)
{
    int8 i, k;

    for (k = 0, i = 0; k < len + (8 - (len % 8)); k += 8, i += 7)
    {
        str[i] = (byte) (((str[k] & 0x7F)) + ((str[k + 1] << 7) &
0x80));

        str[i + 1] = (byte) (((str[k + 1] >> 1) & 0x3F) +
((str[k + 2] << 6) & 0xC0));

        str[i + 2] = (byte) (((str[k + 2] >> 2) & 0x1F) +
((str[k + 3] << 5) & 0xE0));

        str[i + 3] = (byte) (((str[k + 3] >> 3) & 0x0F) +
((str[k + 4] << 4) & 0xF0));

        str[i + 4] = (byte) (((str[k + 4] >> 4) & 0x07) +
((str[k + 5] << 3) & 0xF8));

        str[i + 5] = (byte) (((str[k + 5] >> 5) & 0x03) +
((str[k + 6] << 2) & 0xFC));

        str[i + 6] = (byte) (((str[k + 6] >> 6) & 0x01) +
((str[k + 7] << 1) & 0xFE));
    }

    (*len7bits) = i;
}

/**
    Envoi du SMS

    @param str pointeur vers le texte à envoyer
 */
void
envoyerSMS (char *msg, int8 lenMsg)
{
    int8 k = 0;
    int8 len7bits;

    // Passer le texte en alphabet 7 bits
    conversionAlphabet (msg, lenMsg, &len7bits);

    // Mode PDU
    changerLed ();
    printf ("AT+CMGF=0\r\n");
    delay_ms (500);
    changerLed ();
    // Activer le mode SMS Standard
    printf ("AT+CSMS=0\r\n");
    delay_ms (500);
    changerLed ();

    // Composer le SMS en mode PDU
    printf ("AT+CMGS=");

```

```

// Taille message PDU                                     #BIT BF = 0x94.0
// 21 octets (42 caractères) dans START_PDU + 1 octet de la taille #BIT SSPOV = 0x14.6
printf ("%d", len7bits - 8 + 22);
printf ("\r");
delay_ms (500);

// La conversion hexa doit se faire maintenant et caractère par caractère, l'espace mémoire
// est trop réduit pour traiter une chaîne entière (1 octet -> 2 octets caractères en hexa)
printf (START_PDU); printf ("%2X", lenMsg);
for (k = 0; k < len7bits; k++)
    printf ("%2X", msg[k]);
changerLed ();
// <Ctrl-Z>
printf ("\x1a\r\n");
}
#endif

9.1.2.2. Main.c

/**
 *
 * Carte interface du mobile GSM
 *
 *                               Main.c
 *
 * @version 1.0
 *
 */
#include <16f876.h>
#define icd=true
#define * =16
#define fuses HS,NOWDT,NOPROTECT

#define use delay(clock = 10598400)
#define use rs232(baud = 19200, xmit = PIN_C6, rcv = PIN_C7, parity = n, bits = 8, ERRORS)
#define use i2c(slave, address=0x04, SDA=PIN_C4, SCL=PIN_C3)

// Register SSPSTAT
#define SSPSTAT (*0x94)
// Register SSPBUF
#define SSPBUF (*0x13)
// Register TRISC
#define TRISC (*0x87)
#define PORTB (*0x06)

/** Ajouter 8 zéros est indispensable pour allouer suffisamment d'espace à la chaîne
 * elle est complétée dans conversionAlphabet() pour que le nombre de caractères soit
 * un multiple de 8. Il faut donc allouer suffisamment d'espace pour que l'addition de
 * caractères soit possible.
 */
char sms[] = "Lat: 00N00.0000 \rLon: 000E00.0000 \rAlt 00000m \r00:00:00 Fix:0\0\0\0\0\0\0\0";
const char smsBak[] = "Lat: 00N00.0000 \rLon: 000E00.0000 \rAlt 00000m \r00:00:00 Fix:0\0\0\0\0\0\0\0";
// !! IL N Y'A QUE 7 \0 A LA FIN CAR LE COMPILO RAJOUTE UN \0 DE FIN DE CHAINE !!
// Taille du message utile moins les 8 zéros d'alignement (AVEC LE \0 DE FIN DE CHAINE)
const int8 smsLen = sizeof (sms) - 8;
// Indique l'envoi des données du GPS
int1 gpsLoaded = false;

/**
 * Masquage Data/Address : (val & SSP_DA) 1 si le dernier octet envoyé est une donnée, 0 si c'était l'adresse du composant I2C
 */
#define SSP_DA 0x20

/**
 * Structure d'un message I2C venant de carte GPS
 */

```

```

struct msgI2C
{
    int8 latDeg[2];
    int8 latMin[2];
    int8 latMinDec[4];
    int8 latNS;
    int8 lonDeg[3];
    int8 lonMin[2];
    int8 lonMinDec[4];
    int8 lonEW;
    int8 altitude[5];
    int8 hms[6];
    int8 posValide;
};
/** Variables utilisées par le message SMS */
struct msgI2C valGPS;
/** Mapping structure */
int8 *pValGPS = &valGPS;

/**
 *   Interruption série synchrone
 *   Se déclenche, en mode esclave :
 *       après réception de l'octet d'adresse
 *       après chaque réception de tout octet de donnée
 *
 *   après chaque envoi de tout octet de donnée
 *
 *   *
 *   @see Microchip AN734
 */
#INT_SSP
void
ssp_interupt ()
{
    static int8 k = 0;
    int8 tmp;

    if (i2c_poll ())
    {
        // Si le dernier octet reçu est une adresse
        if (!(SSPSTAT & SSP_DA))
        {
            // réinitialiser l'itérateur
            k = 0;

            tmp = SSPBUF;          // indispensable pour clear du BF
            return;
        }
        else
        {
            pValGPS[k++] = SSPBUF;
            if (k >= sizeof (pValGPS) - 1)
                gpsLoaded = true;
            changerLed();
        }
    }
}

/**
 *   Concaténer n caractères de s2 à la suite de la position de s1
 *   Aucun test de taille.
 *
 *   @param s1 chaine destination
 *   @param s2 chaine à copier
 *   @param idx index dans la chaine destination
 *   @param n nombre de caractères de s2 à copier.
 */
void
strncat (char *s1, char *s2, int8 idx, int8 n)
{
    int k = 0;

    for (k = 0; k < n; k++)
        s1[idx + k] = s2[k];
}

/**
 *   Composer le SMS
 */
void
ecrireSMS (char *sms_)
{
    strncat (sms_, valGPS.latDeg, 5, 2);
    sms_[7] = valGPS.latNS;
    strncat (sms_, valGPS.latMin, 8, 2);
    strncat (sms_, valGPS.latMinDec, 11, 4);
    strncat (sms_, valGPS.lonDeg, 22, 3);
    sms_[25] = valGPS.lonEW;
}

```

```

strncat (sms_, valGPS.lonMin, 26, 2);
strncat (sms_, valGPS.lonMinDec, 29, 4);
strncat (sms_, valGPS.altitude, 39, 5);
strncat (sms_, &valGPS.hms[0], 47, 2);
strncat (sms_, &valGPS.hms[2], 50, 2);
strncat (sms_, &valGPS.hms[4], 53, 2);
sms_[60] = valGPS.posValide;
}

/*
 * Allumer téléphone
 */
void
toggleTel ()
{
    bit_set (PORTB, 1);
    delay_ms (1000);
    bit_clear (PORTB, 1);
    delay_ms (5000);
}

void eteindreTel()
{
    printf("ATE1\x0d");
    delay_ms(100);
    if(kbhit())
        toggleTel();
    // Sinon : téléphone déjà éteint.
    eteindreLed();
}

void allumerTel()
{
    while(kbhit())getch();
    printf("ATE1\x0d");
    delay_ms(1000);
    if(!kbhit())
    {
        toggleTel();
        // Très long temps d'attente, détection réseau, etc...
        delay_ms(10000);
    }
    allumerLed();
    // Sinon : téléphone déjà allumé

```

```

}
/**
    Point d'entrée
 */
void
main ()
{
    int8 k = 60;
    int8 i=0;

    // configuration des périphériques
    enable_interrupts (GLOBAL);
    enable_interrupts (INT_SSP);
    // Led en sortie
    bit_clear (TRISC, 5);
    bit_set (TRISC, 3);
    bit_set (TRISC, 4);
    // Relais en sortie
    set_tris_b (0xFC);
    for( i = 0; i < 10; i++)
    {
        changerLed();
        delay_ms(200);
    }
    eteindreLed ();
    gpsLoaded = false;

    while (1)
    {
        if (gpsLoaded)
        {
            // Envoyer trame

            // Recopier message
            // en effet, envoyerSMS utilise le (char*) sms comme
            variable
            // intermédiaire
            for(i=0; i<sizeof(smsBak); i++)
                sms[i] = smsBak[i];
            // NB: strncat, strncpy ne marchent pas car smsBak est une
            table en ROM
            // et ne peut pas être adressée comme un pointeur avec le
            compilateur PIC-C!!!!

```

```
    gpsLoaded = false;
    allumerTel ();
    delay_ms (4000);
    ecrireSMS (sms);
    envoyerSMS (sms, smsLen);
    delay_ms(5000);
    // Décompte d'une minute
    k = 60;
}
// Une seconde de moins, verrouiller sur les extrêmes.
if (k == 0)
{
    // Eteindre
    eteindreTel ();
    k = 0xFF;
}
else if (k == 0xFF)
{
    k = 0xFF;
}
else
    k--;

delay_ms (1000);
}
}
```

1.1.1.

9.1.3. Brochage Siemens C35

Pin	Name	Function	in/out
1	GND	Ground	
2	SELF-SERVICE	Recognition/control battery charger	in/out
3	LOAD	Charging voltage	In
4	BATTERY	Battery	Out
5	DATA OUT	Data sent	Out
6	DATA IN	Data received	In
7	Z_CLK	Recognition/control accessories	
8	Z_DATA	Recognition/control accessories	
9	MICG	Ground for microphone	In
10	MIC	Microphone input	
11	AUD	Loudspeaker	out
12	AUDG	Ground for external speaker	

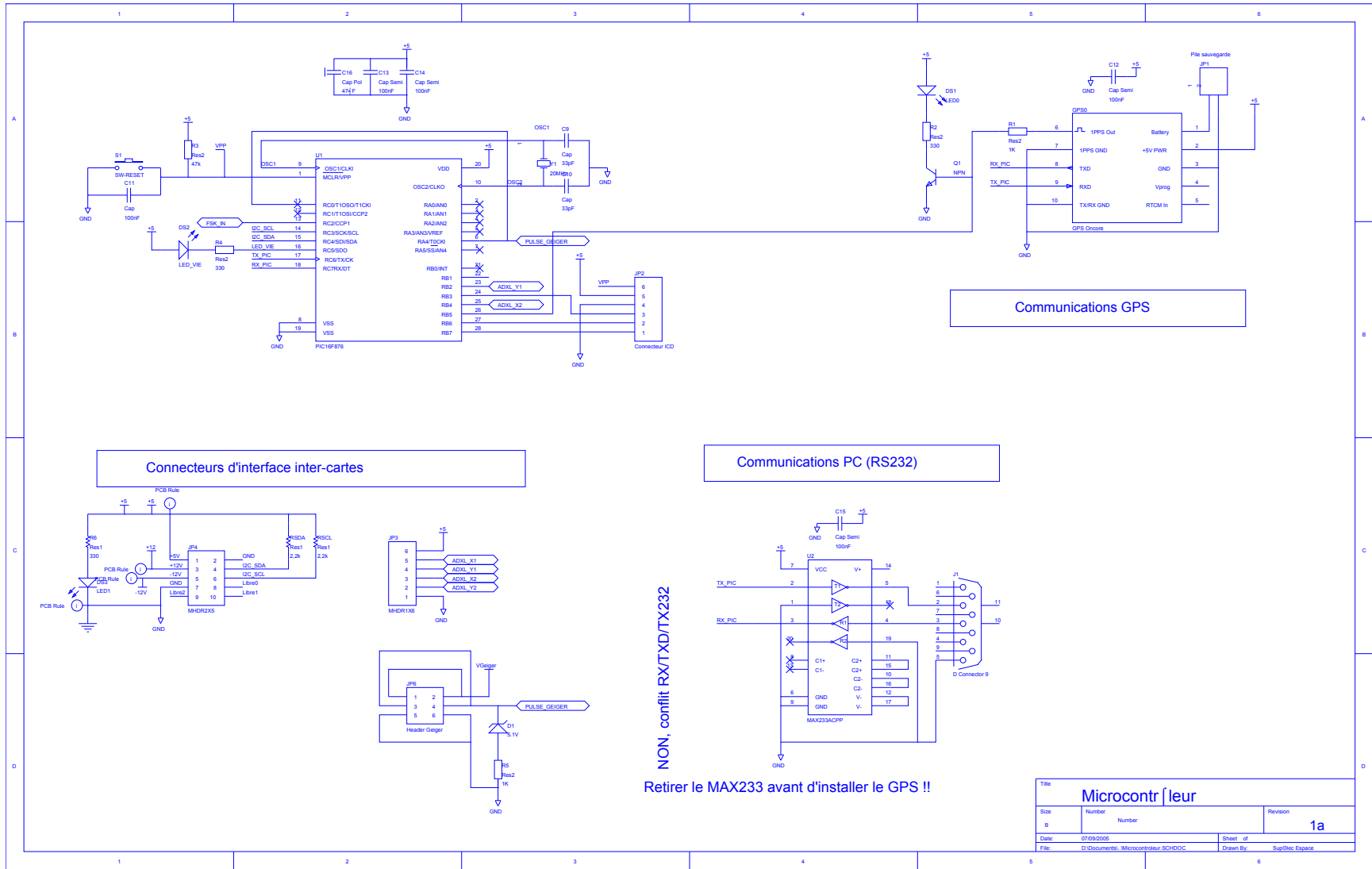
Communications : 19200 bauds, N, 8, 1

9.1.4. Bibliographie

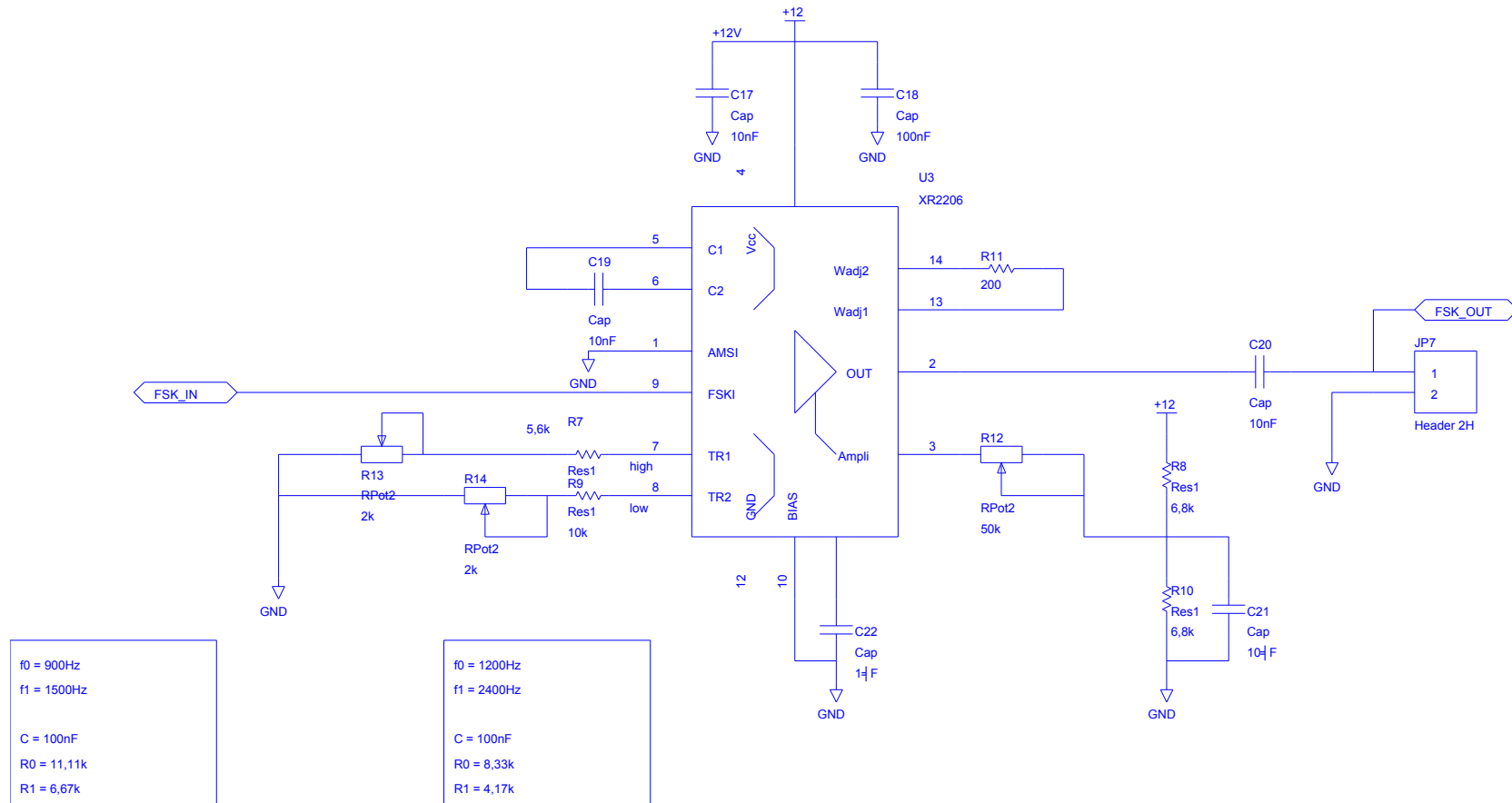
- Technical realization of the Short Message Service (SMS)**, 3rd Generation Partnership Project
Distributed Architecture for applications based on the GSM Short Message Service, G.Martini, G. Rosengo, IEEE 1996
- The Global System for Mobile communications Short Message Service**, tutorial by G. Peersman, S. Cvetkovic, Univ. of Sheffield
- Design of a mobile telecardiology system using GPRS/GSM technology**, M. Elena, J.M. Quero, D.L. Tarrida, L.G. Franquelo, Proceedings of the Seconde Joint EMBS Conference
- Real time stock price distribution utilising the GSM Short Messaging Service**, L. Kilmartin, D. Friel, IEEE 1997
- Appromximate Analytical models for dual band GSM Networks Design and Planning**, M. Meo and A. Marsan
- FH-GSM Frequency Hopping GSM**, .C. Carneheim, S-O.Jonsson, M. Ljungberg
- Channel assignment methods in frequency hopping cellular radio networks**, E. Sarkola
- Electromagnetic fiels radiated by GSM antennas**, M. Bizzi, P. Gianola, Electronics Letters, vol35.11, 1999, p. 855
- Base Station antenna arrays in Mobile communications**, B. Ottersten, P. Zetterberg, Proceedings of 7th International Workshop on Digital Communciations 1995
- Technical Realization of the Point-to-point Short Message Service**, 3GPP Recommandation
- Transmit Power Control in Fixed Cellular Broadband Wireless Systems**, thesis by Salem Salamah, Carleton University, Ottawa, 2000
- The design of location areas in a GSM-network**, pdf
- The effect of Base station antenna height on 900MHz microcellular Mobile radio systems**, J.D. Parsons, A.M.D. Turkami, F. Ju, University of Liverpool
- Ground reflection model**, RF Enginerring for Telecommunications, S. Hudson, Agilent technologies
- Cellular networks optimisation**, Sami Tabbane (ITU), ALTTC
- Wireless Technology : Protocols, Standards, and Techniques**, YACOUB Michel Daoud, 621.384 5 YAC
- Mobile Radio Networks**, WALKER Bernhard H., 621.384 5 WAL
- Principes de radiocommunication de troisième génération : GSM, GPRS, UMTS**, LUCIDARME Thierry, 621.384 5 LUC

The GSM System for Mobile Communications, MOULY Michel, PAUTET Marie-Bernadette, 621.384 5
MOU
Les télécoms mobiles GSM-DCS, SALGUES Bruno, 621.384 5 SAL
Etude de la propagation pour la communication entre mobiles, Bao Yu , 378.242
GSM and UMTS, HILLEBRAND Friedhelm, 621.384 5 HIL
Réseaux GSM, LAGRANGE Xavier / TABBANE Sami / GODLEWSKI Philippe, 621.384 5 LAG
An Introduction To GSM, REDL Siegmund / WEBER Matthias K. / OLIPHANT Malcolm W, 621.384 5
RED

9.2. Carte principale - Télémétries



9.2.2. Schéma Modulateur



9.2.3. Periph_i2c.c

```
#ifndef __PERIPH_I2C_C__
#define __PERIPH_I2C_C__

#define ADDR_I2C_GSM 0x04
#define ADDR_I2C_ADC_ENVIRO 0x54

#include "ledvie.c"
#include "gps.c"
#include "trame.c"

/**
 * Envoi des données vers le GSM.
 * 30 octets de RAM pour tableau.
 * @see utilise les données de la variable globale trameGPS, qui
 doit donc être à jour.
 */
void
envoyerGSM ()
{
    int8 msg[31];
    int8 k;
    int8 i = 0;

    restart_wdt ();

    for (k = 0; k < sizeof (trameGPS.lat); k++)
    {
        if (k != 4) // le '.'
            msg[i++] = trameGPS.lat[k];
    }
    msg[i++] = trameGPS.latNS;

    for (k = 0; k < sizeof (trameGPS.lon); k++)
    {
        if (k != 5)
            msg[i++] = trameGPS.lon[k];
    }
    msg[i++] = trameGPS.lonEW;

    for (k = 0; k < sizeof (trameGPS.altitude); k++)
    {
        msg[i++] = trameGPS.altitude[k];
    }

    for (k = 0; k < sizeof (trameGPS.hms); k++)
    {
        msg[i++] = trameGPS.hms[k];
    }
    msg[i++] = trameGPS.posValide;

    // Start
    i2c_start ();
    // Appel périph
    if (i2c_write (ADDR_I2C_GSM))
    {
        // Problème I2C : esclave inaccessible
        for (k = 0; k < 10; k++)
        {
            delay_ms (200);
            changerLed ();
        }
    }

    for (k = 0; k < sizeof (msg); k++)
        i2c_write (msg[k]);

    // Stop
    i2c_stop ();

    restart_wdt ();

    for (k = 0; k < sizeof (msg); k++)
        fputc (msg[k], FSK);
}

#endif
```

9.2.4. GPS.C

```
#ifndef __GPS_C__
#define __GPS_C__

#include "ledvie.c"

#define SPBRG (*0x99)
#define BRGH = 0x98.2

/** \struct
 * Structure NMEA, en ascii, pas de caractère de terminaison des
 * pointeurs !
 */
struct structNmea
{
    /**> Heures Minutes Secondes hhmmss */
    int8 hms[6];
    /**> Latitude au format ddm. mmmmm */
    int8 lat[9];
    /**> Latitude Nord / Sud */
    int8 latNS;
    /**> Longitude au format dddmm. mmmmm */
    int8 lon[10];
    /**> Longitude Est / Ouest */
    int8 lonEW;
    /**> Position valide */
    int8 posValide;
    /**> Nombre de satellites utilisés */
    int8 activeSat[2];
    /**> HDOP (juste une unité, ça suffira)*/
    int8 hdop;
    /**> Altitude en mètres */
    int8 altitude[5];
};

/**
 * Lire une trame au format ASCII en utilisant une requête @@Eq
 * (protocole binaire)
 * La trame retournée est proche d'une trame NMEA standard, avec
 * des tailles fixes et en 9600 bauds.
 */
* @param nmea Structure structNmea destination du résultat.
* @return false si mauvaise trame en retour
*/
bool
LireGPSEnASCII (struct structNmea *nmea)
{
    const int8 buffer[] = "@@Eq\0\x34\r\n"; // attention ! au
    // \0, pas fin de
    // pointeur !
    int8 k = 0;
    int8 tmp;

    // allumerLed ();
    disable_interrupts (INT_RB);
    restart_wdt ();

    // for(k=0; k<sizeof(buffer) - 1; k++)
    for (k = 0; k < 8; k++)
        fputc (buffer[k], GPS); // pour éviter que puts s'arrête au
    \0

    while (fgetc (GPS) != '@');
    if (fgetc (GPS) != '@')
        return false;
    if (fgetc (GPS) != 'E')
        return false;
    if (fgetc (GPS) != 'q')
        return false;
    tmp = fgetc (GPS); // ", "

    // Sauter mm,dd,yy,
    for (k = 0; k < 9; k++)
        tmp = fgetc (GPS);

    // hh,mm,ss,
    nmea->hms[0] = fgetc (GPS);
    nmea->hms[1] = fgetc (GPS);
    tmp = fgetc (GPS); // ", "
    nmea->hms[2] = fgetc (GPS);
    nmea->hms[3] = fgetc (GPS);
    tmp = fgetc (GPS); // ", "
    nmea->hms[4] = fgetc (GPS);
}
```

```

nmea->hms[5] = fgetc (GPS);
tmp = fgetc (GPS);          // ", "

// Latitude dd,mm.mmmm,n,
nmea->lat[0] = fgetc (GPS);
nmea->lat[1] = fgetc (GPS);
tmp = fgetc (GPS);
for (k = 2; k < 9; k++)
    nmea->lat[k] = fgetc (GPS);
tmp = fgetc (GPS);
nmea->latNS = fgetc (GPS);
tmp = fgetc (GPS);

// Longitude ddd,mm.mmmm,w,
nmea->lon[0] = fgetc (GPS);
nmea->lon[1] = fgetc (GPS);
nmea->lon[2] = fgetc (GPS);
tmp = fgetc (GPS);          // ", "
for (k = 3; k < 10; k++)
    nmea->lon[k] = fgetc (GPS);
tmp = fgetc (GPS);          // ", "
nmea->lonEW = fgetc (GPS);
tmp = fgetc (GPS);          // ", "

// Altitude shhhhh.h
tmp = fgetc (GPS);          // a priori, pas d'altitudes
négatives...
for (k = 0; k < 5; k++)
    nmea->altitude[k] = fgetc (GPS);
tmp = fgetc (GPS);          // "."
tmp = fgetc (GPS);          // dixièmes de mètre
tmp = fgetc (GPS);          // ", "

// Vitesse et azimuth sss.s,hhh.h,
for (k = 0; k < 12; k++)
    tmp = fgetc (GPS);

// m,t,dd.d,
// m: Fix mode
tmp = fgetc (GPS);
tmp = fgetc (GPS);          // ", "
nmea->posValide = fgetc (GPS);
tmp = fgetc (GPS);          // ", "

```

```

// On ne garde que le chiffre des unités du HDOP
tmp = fgetc (GPS);
nmea->hdop = fgetc (GPS);
tmp = fgetc (GPS);
tmp = fgetc (GPS);
tmp = fgetc (GPS);          // ", "
// Nombre de satellites utilisés
nmea->activeSat[0] = fgetc (GPS);
nmea->activeSat[1] = fgetc (GPS);

// Attendre la fin de la trame
while (fgetc (GPS) != '\n');

enable_interrupts (INT_RB);
// eteindreLed ();

restart_wdt ();

return true;
}

/**
 *   Retourne l'altitude de la trame nmea sous forme d'entier.
 *   @return Altitude en mètres
 */
int16
getAltitude (struct structNmea * nmea)
{
    int8 k = 0;
    int16 alt = 0;

    // pour s'amuser, on refait atoi() en l'optimisant pour notre cas
    // particulier.
    for (k = 0; k < sizeof (nmea->altitude); k++)
        alt = 10 * alt + (nmea->altitude[k] - '0');

    return alt;
}

/**
    Passer l'USART en 9600 bauds

```

```

    @see L'utilisation des #use rs232... n'est pas très fiable
pour les changements de vitesse.
*/

```

```

void
set9600Bauds ()
{
    BRGH = 1;
    SPBRG = (FOSC / 9600 / 16) - 1;
}

```

```

/**
    Passer l'USART en 4800 bauds

```

```

    @see L'utilisation des #use rs232... n'est pas très fiable
pour les changements de vitesse.
*/

```

```

void
set4800Bauds ()
{
    BRGH = 0;
    SPBRG = (FOSC / 4800 / 64) - 1;
}

```

```

/**
    Passer le GPS en mode NMEA

```

```

*/
void
gpsMotorolaToNMEA ()
{
    set9600Bauds ();

    fputs ("@@Ci\\1\\x2b\\r\\n", GPS);

    set4800Bauds ();

```

```

    delay_ms (500);          // Au moins, sinon les requêtes NMEA // / machine d'état
ne
    // sont pas prises en compte.
    delay_ms (500);
}

```

```

#endif // __GPS_C__

```

9.2.5. Main.c

```

/** \file
 *   Gestion des téléméasures, GPS et bus I2C
 *
 *
 *
 */

#include <16f876a.h>
#define *=16
#define opt 9
#define FOSC 3686400
#define PORTB (*0x06)
#define use delay(clock=FOSC)

#define use rs232(baud=600, xmit=PIN_C2, parity=n, bits=8, stream=FSK,
ERRORS)
#define use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, parity=n, bits=8,
stream=GPS, ERRORS)

#define use i2c(master, SDA=PIN_C4, SCL=PIN_C3)

#define bool int1

#include "ledvie.c"
#include "timer.c"
#include "gps.c"
#include "trame.c"
#include "periph_i2c.c"

#define ETAT_IDLE 0
#define ETAT_ENVOYER_GPS 0x01
#define ETAT_ENVOYER_STATUT 0x02
#define ETAT_ENVOYER_ENVIRO 0x04
#define ETAT_GEIGER 0x08
#define ETAT_ENVOYER_GSM 0x10
*/

```

```

* t Séquenceur Action 0 0 GSM (+GPS) 1 (GPS) * interruption
masquée... 2
* 1 3 2 ENVIRO & RAZ_GEIGER 4 3 5 4 GPS 6 (GPS) 7 5 8 6 ENVIRO &
* RAZ_GEIGER 9 7
*
* NB : le GSM et le GPS sont liés car le GSM à besoin d'une trame
GPS
* récente.
*
*/

/*
* Temps mort : 190µs ==> 5263 cps maxi
*/
int8 seqEtats[] =
{
    ETAT_GEIGER | ETAT_ENVOYER_ENVIRO,
    ETAT_ENVOYER_GPS,ETAT_ENVOYER_STATUT,
    ETAT_GEIGER | ETAT_ENVOYER_ENVIRO,
    ETAT_ENVOYER_GPS,ETAT_ENVOYER_STATUT,
    ETAT_GEIGER | ETAT_ENVOYER_ENVIRO,
    ETAT_ENVOYER_GPS,ETAT_ENVOYER_STATUT,
    ETAT_GEIGER | ETAT_ENVOYER_ENVIRO,
    ETAT_ENVOYER_GPS,ETAT_ENVOYER_GSM
    /*
    *      0      1      2      3      4
    *      G+E  GPS----- STATUT
    */
};

volatile int8 indexEtat = 0;
bool flagNextStep = true;

/**
* Initialisations périphériques et variables
*/
void
initialisations ()
{
    int k;
    restart_wdt ();
    setup_wdt (WDT_2304MS);
    restart_wdt ();

    set_tris_b (0xFF);
    set_tris_c (0xD3);
    set9600Bauds ();

    // Délai important pour la mise en route du module GPS.
    // 12*250ms = 3s
    for (k = 0; k < 12; k++)
    {
        delay_ms (250);
        restart_wdt ();
        changerLed ();
    }
    eteindreLed ();

    // Configuration Timer0 (compteur Geiger)
    initTimer1 ();

    // Autoriser les interruptions
    enable_interrupts (GLOBAL);
    enable_interrupts (INT_RB);
    enable_interrupts (INT_TIMER1);
    // Variables
    trameStatut.nbSMS = 0;
    trameEnviro.geiger = 0;
}

/*!
    Interruption sur changement de RB4 à RB7
*/
#int_rb
intPortB ()
{
    static int8 compteur = 0;

    // Vérifier que l'interruption vient bien de la ligne 1PPS du GPS
    // lecture sur le port B nécessaire pour clear du flag
    d'interruption
    if (!(input_b () & 0x20))
        return;

    // Incrémenter la machine d'état

```

```

if (++indexEtat > sizeof (seqEtats))
    indexEtat = 0;
flagNextStep = true;
if (seqEtats[indexEtat] & ETAT_GEIGER)
{
    changerLed();
    // Genre d'optimisation simple mais qui passe mal
    trameEnviro.geiger = get_timer1();
    set_timer1(0);
}
}

/*!
 *   Mode de sécurité. Si un redémarrage sur watchdog a eu lieu,
 *   c'est qu'il
 *   n'y a pas eu d'interruption de l'horloge GPS pour plus de
 *   2secondes, donc
 *   une possible perte de l'horloge du séquenceur.
 */
void
modePanic ()
{
    /*
     * Envoyer les télémessures, envoyer un SMS de détresse
     */
}

/*!
 *   Algo se basant sur l'altitude donnée par le GPS pour déterminer
 *   si l'envoi
 *   d'un SMS doit avoir lieu.
 */
bool
testEnvoiGSM ()
{
    unsigned int16 alt;

    alt = getAltitude (&trameGPS);

    if (alt < 1024)
        return true;
    else
        return false;
}

/**
 *   Point d'entrée
 */
void
main ()
{
    initialisations ();

    // test du watchdog, obligatoirement APRES l'initialisation de
    // nbResetWdt à 0 (dans intilisations())
    if (restart_cause () == WDT_TIMEOUT)
    {
    }
    while (1)
    {
        restart_wdt ();
        if (flagNextStep)
        {
            flagNextStep = false;
            switch (seqEtats[indexEtat])
            {
                case ETAT_GEIGER | ETAT_ENVOYER_ENVIRO :
                case ETAT_ENVOYER_ENVIRO:
                    envoyerTrameEnviro ();
                    break;

                case ETAT_ENVOYER_GPS:
                    envoyerTrameGPS ();
                    break;

                case ETAT_ENVOYER_STATUT:
                    envoyerTrameStatut ();
                    break;

                case ETAT_ENVOYER_GPS | ETAT_ENVOYER_GSM:
                case ETAT_ENVOYER_GSM :

```

```

    if (testEnvoiGSM())
    {
        trameStatut.nbSms++;
        envoyerGSM ();
    }
    break;

case ETAT_IDLE:
default:
    break;
}
}
}
}

```

9.2.6. Trame.C

```

/*****
*****
*      Création de la trame de télémessure
*
*
*
*****
*****/
#ifndef __TRAME_C__
#define __TRAME_C__

#include "gps.c"
#include "ledvie.c"
#include "max127.c"
#include "timer.c"
#include "tmp100.c"
#define ID_TMP_INT 0 /* adresse du capteur relié à la masse
1001000 */
#define ID_TMP_EXT 6 /* adresse du capteur relié au +5 1001110 */
#define ID_ADC ALIM 1
/* Pour utilisation avec l'ancienne carte à MAX127
define ENVIRO_VOIE_PRESSION 0
#define ENVIRO_VOIE_TEMP_INT1 1
#define ENVIRO_VOIE_TEMP_INT2 2
#define ENVIRO_VOIE_TEMP_EXT3
*/
#define TYPE_TRAME_GPS 0x91
#define TYPE_TRAME_EXPERIENCE 0x13
#define TYPE_TRAME_STATUT 0xB4

const int8 preambule[3] = {
    0xAA, 0xFF, 0xAA
};

/** \struct
*      Données position GPS (sur la base d'une trame NMEA)
*/

```

```

struct structNmea trameGPS;

/** \struct
 *      Données d'expérience.
 */
struct sDonneesExperiences
{
    int8 tempeInt;           // /< Température intérieure
    int8 tempeExt;          // /< Température extérieure
    int16 pression;         // /< Pression Pa
    int16 geiger;           // /< Nb de coups au compteur
};

struct sDonneesExperiences trameEnviro;

/** \struct
 *      Etat du ballon
 */
struct sStatutBallon
{
    int8 tension1;
    int8 tension2;
    int8 tension3;
    int8 nbSMS;
};

struct sStatutBallon trameStatut;

/**
 *      Envoyer une trame. La fonction envoie le préambule de
synchronisation,
 *      l'octet de type de trame <code>typeTrame</code>, les données
et la
 *      checksum. La checksum est calculée comme étant le XOR des
données
 *      envoyées.
 *
 *      @param data Suite de données à envoyer.
 *      @param taille Nombre d'octets à envoyer
 *      @param typeTrame Octet codant le type de trame à envoyer.
 */
bool
envoyer (int8 * data, int8 taille, int8 typeTrame)
{
    int8 k = 0;
    int8 checksum = 0;

    disable_interrupts (INT_RB);

    for (k = 0; k < sizeof (preambule); k++)
        fputc (preambule[k], FSK);

    fputc (typeTrame, FSK);

    for (k = 0; k < taille; k++)
    {
        checksum ^= data[k];
        fputc (data[k], FSK);
    }

    fputc (checksum, FSK);
    enable_interrupts (INT_RB);
}

/**
 *      Créer et envoyer la trame GPS
 */

bool
envoyerTrameGPS ()
{
    lireGPSEnASCII (&trameGPS);
    // pour la renvoyer au sol
    envoyer ((int8 *) & trameGPS, sizeof (trameGPS), TYPE_TRAME_GPS);
}

/**
 *      Créer et envoyer la trame décrivant les résultats des
expériences
 *      environnement.
 */
void
envoyerTrameEnviro ()
{

```

```

trameEnviro.tempeInt =
    lireTmp100 (ID_TMP_INT) >> 8;
trameEnviro.tempeExt =
    lireTmp100 (ID_TMP_EXT) >> 8;

// la valeur du compteur Geiger est initialisée depuis la fonction
// timer
envoyer ((int8 *) & trameEnviro, sizeof (trameEnviro),
        TYPE_TRAME_EXPERIENCE);
}

/**
 * Envoyer trame de statut
 */
void
envoyerTrameStatut ()
{
    trameStatut.tension1 = lireAdcMax127 (ID_ADC_ALIM, 0) >> 8;
    trameStatut.tension2 = lireAdcMax127 (ID_ADC_ALIM, 1) >> 8;
    trameStatut.tension3 = lireAdcMax127 (ID_ADC_ALIM, 2) >> 8;
    envoyer((int8*) &trameStatut,
sizeof(trameStatut),TYPE_TRAME_STATUT);
}

#endif // __TRAME_C__

```

9.3. Alimentations

