



03 JANVIER 2019

RAPPORT ADS-B

SECURITE DES OBJETS MOBILES COMMUNIQUE

MAHAMAN BACHIR ATTOUMAN OUSMANE

UPHF VALENCIENNES

Master CDSI




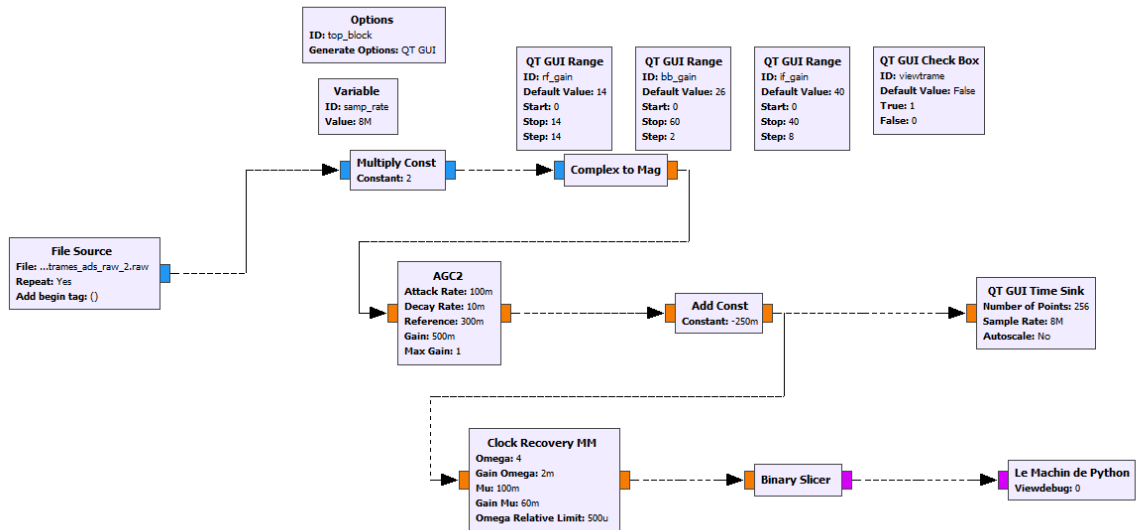
Table des matières

1.	Construction des blocs	2
2.	Exécution	2
3.	Lecture et compréhension du code python :	3
3.1.	Sélection de la trame.....	3
3.2.	Reconstruction de la trame au format standard.....	3
3.3.	Trame de taille 112.....	3
3.4.	Trame de taille 56.....	4
4.	Décoder la position de l'avion	5
4.1.	Recherche des bits indiquant la latitude et la longitude.....	5
4.2.	Calcul de la latitude	5
5.	Gestion du Parity Check :	6
4.1	Ecriture de la fonction	6
4.2	Appel de la fonction	6

Rapport ADS-B

1. Construction des blocs

Nous reproduisons exactement les blocs donnés en cours et changeons **osmocom source** par **file source**, nous y indiquons le chemin des captures données en cours. Pour le dernier bloc nous écrivons aussi le script python donné en cours.



Résultat :

Fig.1 Blocs Gnu-Radio

2. Exécution

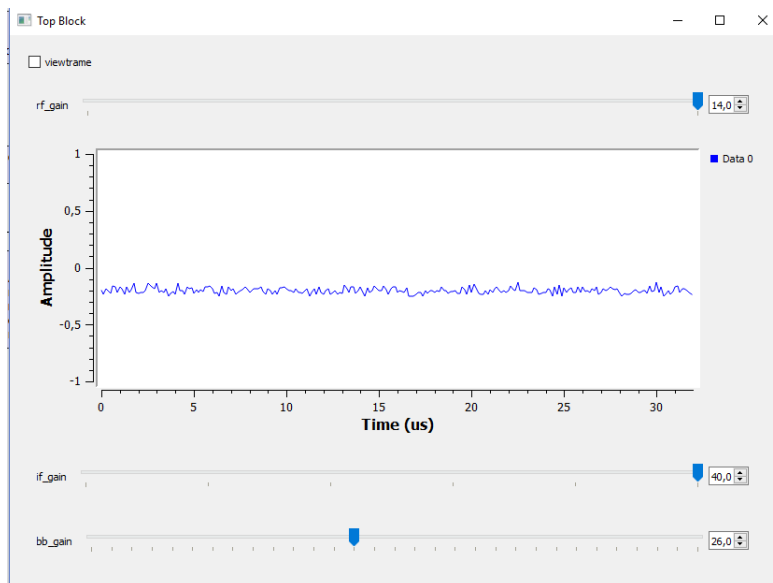


Fig.2 Scope

```

11 0x424117 10957.56 m
19 0x424117 vitesse = 876 km/h cap = 195 deg
19 0x424117 vitesse = 876 km/h cap = 195 deg
11 0x424117 10934.7 m
19 0x471f52 vitesse = 892 km/h cap = 103 deg
19 0x471f52 vitesse = 892 km/h cap = 103 deg
19 0x471f52 vitesse = 892 km/h cap = 103 deg
11 0x424117 10858.5 m
12 0x471f52 10652.76 m
11 0x424117 10843.26 m
19 0x471f52 vitesse = 892 km/h cap = 103 deg
11 0x424117 10843.26 m
19 0x424117 vitesse = 866 km/h cap = 199 deg
19 0x424117 vitesse = 866 km/h cap = 199 deg
19 0x424117 vitesse = 866 km/h cap = 199 deg
19 0x424117 vitesse = 866 km/h cap = 199 deg
11 0x424117 10972.8 m
19 0x424117 vitesse = 865 km/h cap = 199 deg
12 0x4ca1d0 11277.6 m
12 0x471f52 10668.0 m
19 0x471f52 vitesse = 890 km/h cap = 103 deg
19 0x424117 vitesse = 879 km/h cap = 194 deg
19 0x424117 vitesse = 879 km/h cap = 193 deg
12 0x471f52 10660.38 m
19 0x471f52 vitesse = 892 km/h cap = 103 deg
19 0x471f52 vitesse = 892 km/h cap = 103 deg
19 0x424117 vitesse = 866 km/h cap = 199 deg
19 0x424117 vitesse = 866 km/h cap = 199 deg
11 0x424117 10972.8 m

```

Fig.3 Données issues des trames

3. Lecture et compréhension du code python :

3.1. Sélection de la trame

Analysons d'abord la fonction « Work » :

Le bloc python reçoit des données binaires initialement dans « input items [0] »

Ces données sont copiées dans notre variable « trame ». Ensuite il cherche les trames non vides. (Avec (np. Where ==1) [0]).

Nous vérifions les 14 premiers bits (le préambule dans « sr ») pour savoir s'il s'agit d'une trame ADS-B, bien synchronisée. Si c'est le cas la trame est décodée, le préambule sera pas la suite « enlevé ».

3.2. Reconstruction de la trame au format standard

En fait sur chaque 2 bits les données utiles se trouvent sur le MSB (à gauche), le deuxième étant toujours l'opposé du précédent, si on a **100110101101** la trame sera constituée des bits en rouge.

Ensuite selon la taille de la trame elle est décodée selon l'énoncé du standard.

3.3. Trame de taille 112

Pour décoder la trame la première chose à faire était de vérifier que le champ Downlink Format est égal à 17, identifiant des communications civile ADS-B. Ces trames avec df = 17 peuvent alors être décodées. Les champs sont convertis d'abord en décimal avant traitement.

- Nous avons ensuite les identifiants uniques à chaque aéronef

-Les bits 32 à 37 (Tc) donne le code qui définit le type de données contenues dans la suite des données de la trame.

Ensuite avec un tableau de format type code (tel que celui-ci-dessous **fig1.**), Nous somme alors capable de savoir quelles données sont situés à quel position dans la trame

3.4. Trame de taille 56

Le même procédé est appliqué. Il s'agit de lire la documentation explicitant le format des trames et de le traduire en code.

TABLE 4 – Tableaux des valeurs du FTC pour $FTC \in [0, 8]$

TYPE Code	Format	Horizontal protection limit (HPL)	95% Containment radius, μ and ν , on horizontal and vertical position error	Altitude type (see §A.2.3.2.4)	NUC _r
0	No position information			Barometric altitude or no altitude information	0
1	Identification (Category Set D)			Not applicable	
2	Identification (Category Set C)			Not applicable	
3	Identification (Category Set B)			Not applicable	
4	Identification (Category Set A)			Not applicable	
5	Surface position	HPL < 7.5 m	$\mu < 3$ m	No altitude information	9
6	Surface position	HPL < 25 m	$3 \text{ m} \leq \mu < 10$ m	No altitude information	8
7	Surface position	HPL < 185.2 m (0.1 NM)	$10 \text{ m} \leq \mu < 92.6$ m (0.05 NM)	No altitude information	7
8	Surface position	HPL > 185.2 m (0.1 NM)	(0.05 NM) $92.6 \text{ m} \leq \mu$	No altitude information	6

TABLE 5 – Tableaux des valeurs du FTC pour $FTC \in [9, 18]$

9	Airborne position	HPL < 7.5 m	$\mu < 3$ m	Barometric altitude	9
10	Airborne position	$7.5 \text{ m} \leq \text{HPL} < 25$ m	$3 \text{ m} \leq \mu < 10$ m	Barometric altitude	8
11	Airborne position	$25 \text{ m} \leq \text{HPL} < 185.2$ m (0.1 NM)	$10 \text{ m} \leq \mu < 92.6$ m (0.05 NM)	Barometric altitude	7
12	Airborne position	$185.2 \text{ m} (0.1 \text{ NM}) \leq \text{HPL} < 370.4$ m (0.2 NM)	$92.6 \text{ m} (0.05 \text{ NM}) \leq \mu < 185.2$ m (0.1 NM)	Barometric altitude	6
13	Airborne position	$370.4 \text{ m} (0.2 \text{ NM}) \leq \text{HPL} < 926$ m (0.5 NM)	$185.2 \text{ m} (0.1 \text{ NM}) \leq \mu < 463$ m (0.25 NM)	Barometric altitude	5
14	Airborne position	$926 \text{ m} (0.5 \text{ NM}) \leq \text{HPL} < 1852$ m (1.0 NM)	$463 \text{ m} (0.25 \text{ NM}) \leq \mu < 926$ m (0.5 NM)	Barometric altitude	4
15	Airborne position	$1852 \text{ m} (1.0 \text{ NM}) \leq \text{HPL} < 3704$ m (2.0 NM)	$926 \text{ m} (0.5 \text{ NM}) \leq \mu < 1852$ m (1.0 NM)	Barometric altitude	3
16	Airborne position	$3704 \text{ km} (2.0 \text{ NM}) \leq \text{HPL} < 18.52$ km (10 NM)	$1.852 \text{ km} (1.0 \text{ NM}) \leq \mu < 9.26$ km (5.0 NM)	Barometric altitude	2
17	Airborne position	$18.52 \text{ km} (10 \text{ NM}) \leq \text{HPL} < 37.04$ km (20 NM)	$9.26 \text{ km} (5.0 \text{ NM}) \leq \mu < 18.52$ km (10.0 NM)	Barometric altitude	1
18	Airborne position	$\text{HPL} \geq 37.04$ km (20 NM)	$18.52 \text{ km} (10.0 \text{ NM}) \leq \mu$	Barometric altitude	0

Fig1. Tableau de format type code

4. Décoder la position de l'avion

4.1. Recherche des bits indiquant la latitude et la longitude

Nous cherchons d'abord quel est le type code de la position :

Il s'agit de TC = [9 à 19]. A l'aide du document fourni « ADS-B for dummies »

Number of bits	Contents
5	Format type code
2	Surveillance status
1	Single antenna flag
12	Altitude
1	Time
1	CPR format
17	CPR encoded latitude
17	CPR encoded longitude
56 bits total	

On sait que les données commencent à partir du 32^{ème} bit

Donc l'altitude est comprise entre $32 + 8 = 40$ et $40 + 12 - 1 = 51^{\text{ème}}$ bits dans notre liste S [40 :52]

De meme la latitude est située entre le 54^{ème} et le $54 + 17 - 1 = 70^{\text{ème}}$ bits soit S [54 :71] en code python

Aussi la longitude est située entre le 71^{ème} et le $71 + 17 - 1 = 87^{\text{ème}}$ bits soit S [71 :88] en python

4.2. Calcul de la latitude

Pour trouver la latitude nous devons effectuer d'abord certains calculs à partir des données brutes , la procédure que nous avons suivi réf : <https://mode-s.org/decode/adsb/airborne-position.html> , cependant nous traiterons uniquement la latitude nord ,

Étant donné que le but du travail est de comprendre les procédures type pour le décodage des trames. Pour la longitude une démarche similaire est à suivre.

```
i = int(S[53],2)
lat= int(S[54:71],2)
lon = int(S[71:88],2)
lat_cpr_even = lat /131072
lon_cpr_even = lon/131072
lat_cpr_odd = lat / 131072
lon_cpr_odd = lon/131072
dlat_even = 90 /60
dlat_odd = 90/59
j = floor (59 * lat_cpr_even - 60 * lat_cpr_odd + 0.5)
latitude = float(dlat * (j%60 + lat_cpr_even))
print tc, " ",hex(ica024), "latitude = ",latitude," deg Nord"
```

Nous plaçons le code juste au-dessous du calcul de l'altitude.

5. Gestion du Parity Check :

Le Parity check permet de savoir si les données ont été altérées durant la transmission. Pour l'ADS-B le Parity check se trouve sur les 24 derniers bits de la trame donc S [-24 :]. Le polynôme générateur est donné sur le site officiel de ADS-B, ainsi que l'algorithme de vérification du Parity check que nous adaptons en python. Comme résultat si le CRC = 0 la trame est intègre sinon altérée alors pas besoin de la traiter. Notre code fonctionne de la façon suivante « if check(trame) : la trame est bonne »

4.1 Ecriture de la fonction

```
#-----Calcul CRC-----
def check(self,t):
    generator = "111111111111010000001001"
    t = list(t)
    for i in range(0,len(t)-24):
        if t[i] == "1" :
            for j in range(len(generator)):
                t[i+j] =str( (int(t[i+j]) ^ int(generator[j])) )
    t=''.join(t[-24:])
    if t == "0"*24 :
        if self.viewDebug:
            print t , "-*-*-*-*-*-*-*-*-*-*-*"
        return True
    return False
#-----Calcul CRC -----
```

Fig. Parity check

4.2 Appel de la fonction

Avant de décoder une trame nous vérifions d'abord qu'elle est intègre sinon on passe à la trame suivante. Nous modifierons alors la fonction `decodetrame()`. Nous y insérons l'appel de notre fonction `check()` juste avant le décodage des trames.

```

        break
    if self.trame[0] == 1:
        S = S + "1"
    else:
        S = S + "0"
    self.trame = self.trame[2:]
if self.check(S):

    if len(S) == 112:

        self.decodetrame112(S)

    if len(S) == 56:

        self.decodetrame56(S)
else:
    if self.viewDebug:
        print "trame alteree: " , S
    if len(S) != 112:

```