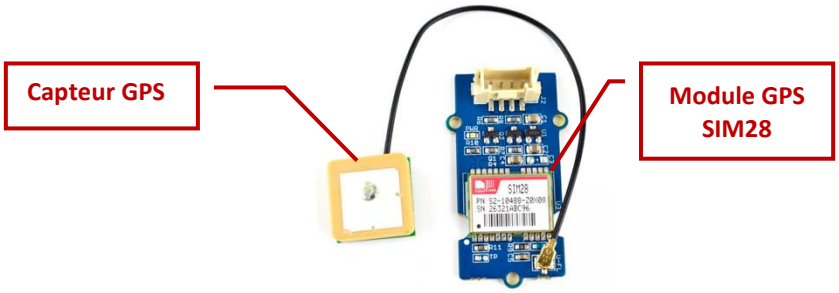
STI2D – Enseignement de spécialité SIN

SHIELD GPS SEEED

1 – MODULE GPS SEN10752P

1.1 – Présentation

Le module GPS **SEN10752P** du fabricant SEEED est un shield équipé d'un capteur GPS "E-1612-UB" et d'un module GPS SIM28 compatible avec le protocole NMEA et le protocole propriétaire U-blox 6. La communication avec module est réalisée au moyen d'une **liaison série asynchrone** (UART).



1.2 – Caractéristiques

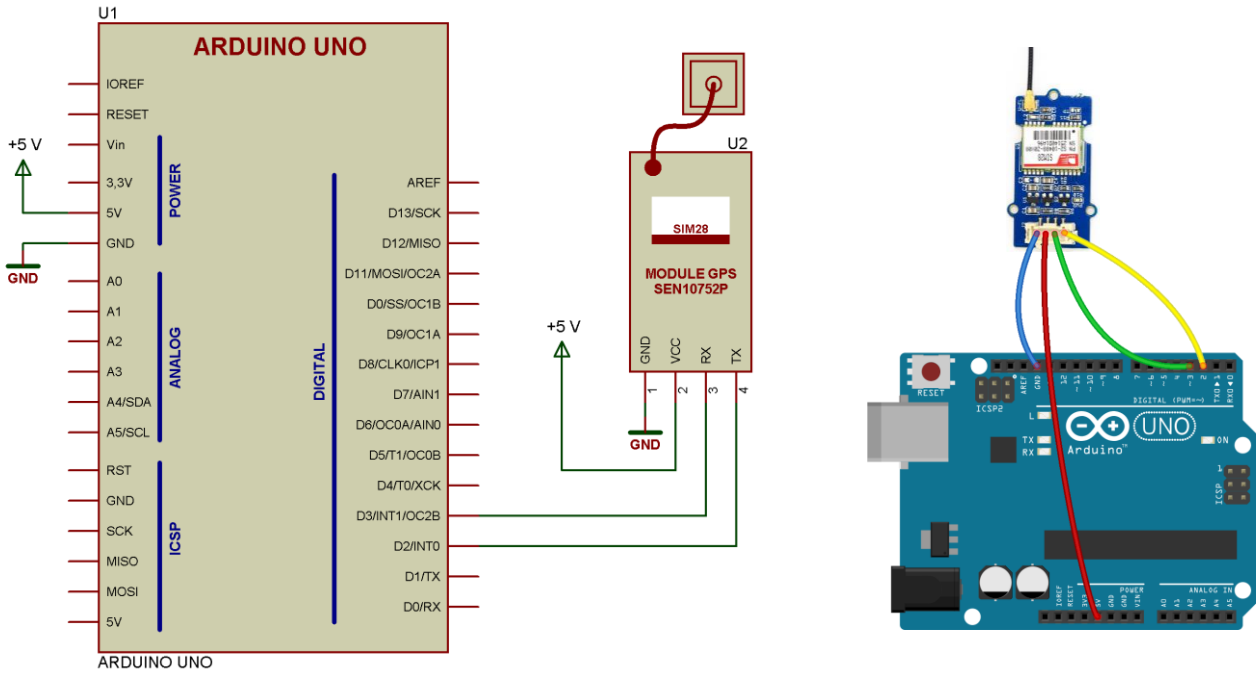
Caractéristiques	Valeurs
Tension d'alimentation	3,3 V ou 5 V
Consommation	40 mA sous 3,3 V
Sensibilité	-160 dBm
Vitesse de transmission	4800 bits/s (version SIM28)
Dimensions shield	40 x 20 x 13 mm
Dimensions capteur GPS	15 x 15 x 7 mm
Température de fonctionnement	– 40 °C à + 85 °C
Protocoles	NMEA, U-blox 6

2 – UTILISATION DU MODULE GPS SEN10752P AVEC LA CARTE ARDUINO

2.1 – Schéma de câblage

Le shield GPS est alimenté sous 5 V. La liaison série entre la carte arduino et le shield GPS sera réalisée par une **liaison série logicielle** sur les broches numériques **D2** (**Rx**) et **D3** (**Tx**) de la carte Arduino.

La carte Arduino ne va que recevoir des données issues du shield GPS. La connexion entre la broche Rx du shield et la broche numérique D3 n'est pas nécessaire.



2.2 – liaison série asynchrone logicielle

L'Atmega328 possède une **interface de communication série** UART accessible, grâce aux **broches numériques 0** (**Rx**) et 1 (**Tx**). La bibliothèque « **SoftwareSerial** » a été développée pour permettre la communication série sur d'autres broches numériques de l'Arduino de manière logicielle. Il est possible de gérer plusieurs ports séries logiciels avec des vitesses allant jusqu'à **115 200 bps** cependant un seul peut recevoir des données à la fois.

Inclure la bibliothèque « SoftwareSerial »

Pour inclure la librairie « SerialSoftware » dans un programme, on ajoutera au début du programme la ligne suivante :

```
#include <SoftwareSerial.h>
```

Créer une liaison série logicielle

```
SoftwareSerial Nom_Liaison(Broche_Rx, Broche_Tx)
```

Exemple : Création d'une liaison série logicielle nommée GPS sur les broches 2 (Rx) et 3 (Tx) :

```
SoftwareSerial GPS(2, 3)
```

Fonction « begin »

```
Nom_Liaison.begin(Vitesse) ;
```

Cette fonction qui doit être **appelée au moins une fois**, généralement dans la fonction setup(), permet de **définir la vitesse** utilisée par la liaison série.

La valeur prise par la variable « **Vitesse** » doit être une des **vitesses définies par la norme RS232**.

Exemple :

```
GPS.begin(115200);
```

Fonction « available »

```
Nom_Liaison.available() ;
```

Cette fonction permet de connaitre de nombre d'octet reçus sur l'entrée Rx et stockés dans la mémoire tampon du registre de réception.

Exemple :

```
N_octet = GPS.available();
```

Fonction « read »

```
Nom_Liaison.read() ;
```

Cette fonction permet de lire les octets reçus sur l'entrée Rx et stockés dans la mémoire tampon du registre de réception.

Exemple :

```
Cmd = GPS.read();
```

Fonctions « print » et « println »

```
Nom_Liaison.print(donnee) ;
```

La fonction « print » permet de transmettre une donnée sur la sortie Tx. La fonction « println » permet de transmettre une donnée suivie d'un retour chariot et saut de ligne.

Exemple :

```
GPS.print('Test');
```

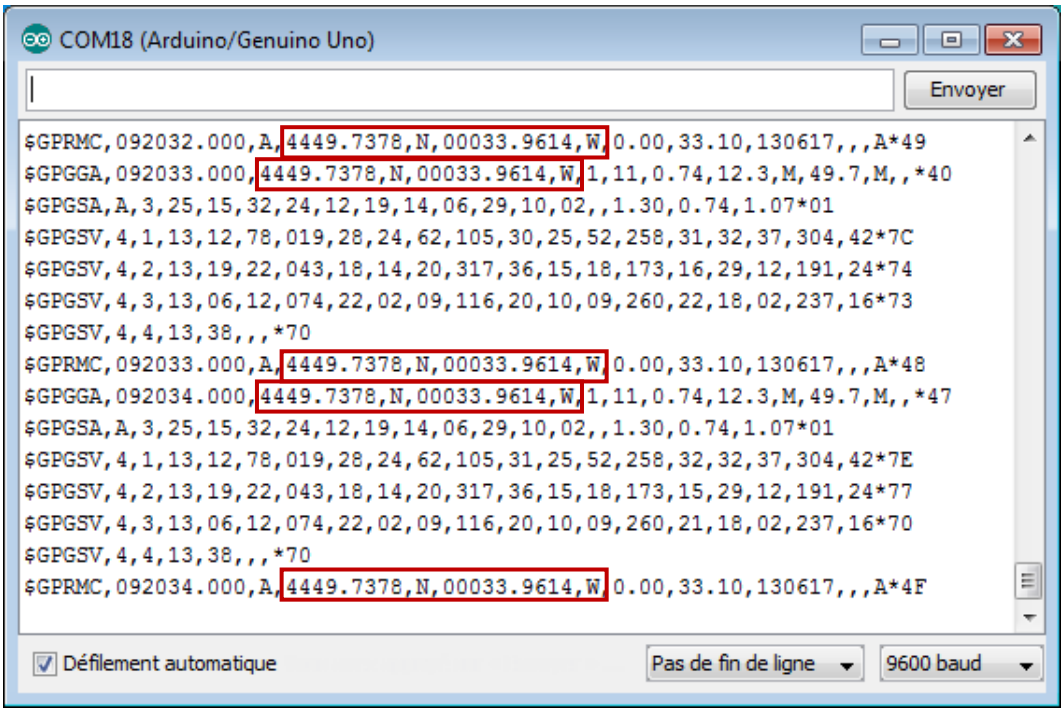
2.3 – Programme

Le programme suivant permet de lire l'ensemble des trames transmises par le shield GPS et de les afficher sur le moniteur série.

Programme Arduino

```
#include <SoftwareSerial.h>

SoftwareSerial SoftSerial(2, 3);            // Port série logiciel Rx = 2 et Tx = 3
unsigned char buffer[64];                   // Tableau de 64 caractère contenant les données reçues par la liaison série
int count = 0;                              // Indice permettant de parcourir le tabelau buffer

void setup() {
  SoftSerial.begin(9600);                 // Vitesse de commnunication du port série logiciel
  Serial.begin(9600);                     // Vitesse de commnunication du port série matériel (UART)
}

void loop() {
  if (SoftSerial.available()) {               // Si une donnée a été reçue par le port série logiciel
    while (SoftSerial.available())  {       // Tant que des données sont disponible sur le port série logiciel
      buffer[count++] = SoftSerial.read();  // Lire les données du port série et les écrire dans le tableau buffer
      if (count == 64) break;               // Si le nombre de caractéres atteint 64 arrêter la lecture
    }
    Serial.write(buffer, count);               // Afficher dans le moniteur serie les données contenues dans le tableau buffer
    clearBufferArray();                        // Appel de la fonction ClearBufferArray() qui efface le contenu du tableau buffer
    count = 0;                                 // palcer l'indice de comptage à 0
  }
}



void clearBufferArray()                       // fonction qui efface le contenu du tableau buffer
{
  for (int i = 0; i < count; i++)
  {
    buffer[i] = NULL;
  }
}
```

Résultats moniteur série

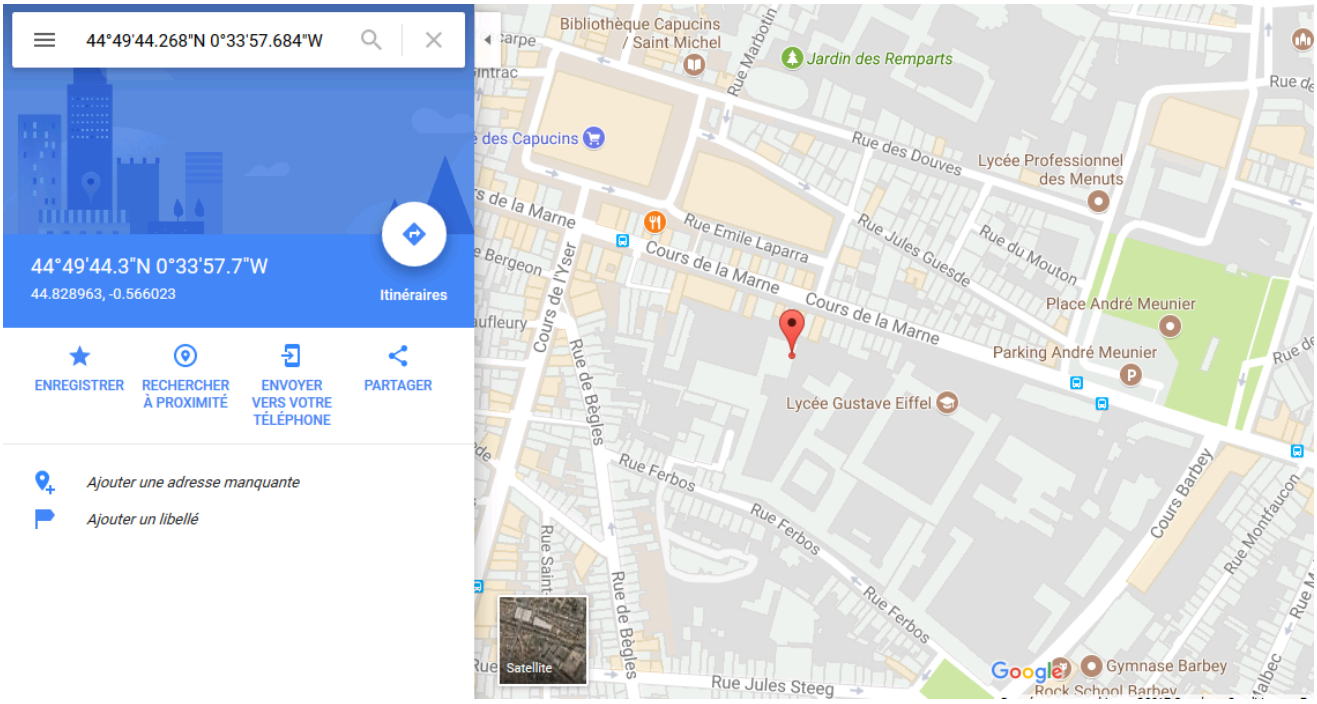


<u>Coordonnées</u> :

Latitude : **4449.7378,N = 44,829° Nord = 44° 49,7378′ Nord = 44° 49′ 44,268 ″ Nord**

Longitude **00033.9614,W = 0,566° Ouest = 0° 33,9614′ Ouest = 0° 33′ 57,684 ″ Ouest**

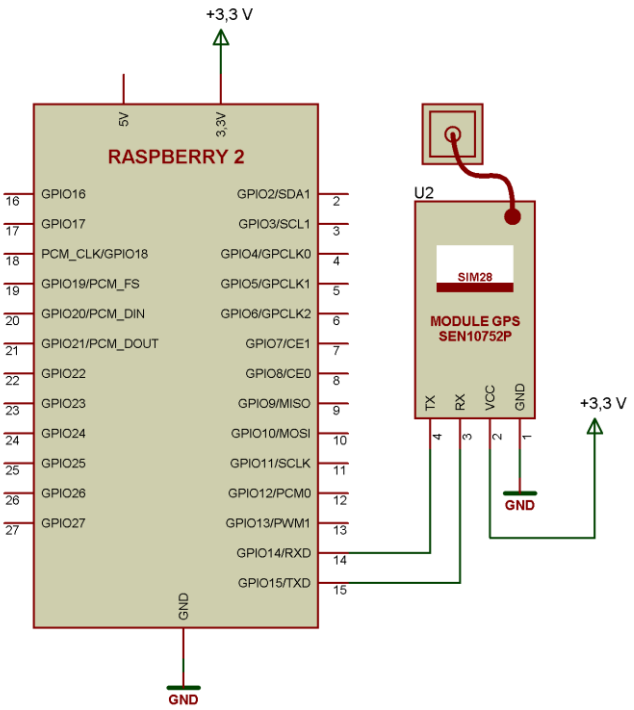
Vérification du résultat sur « Google Maps »

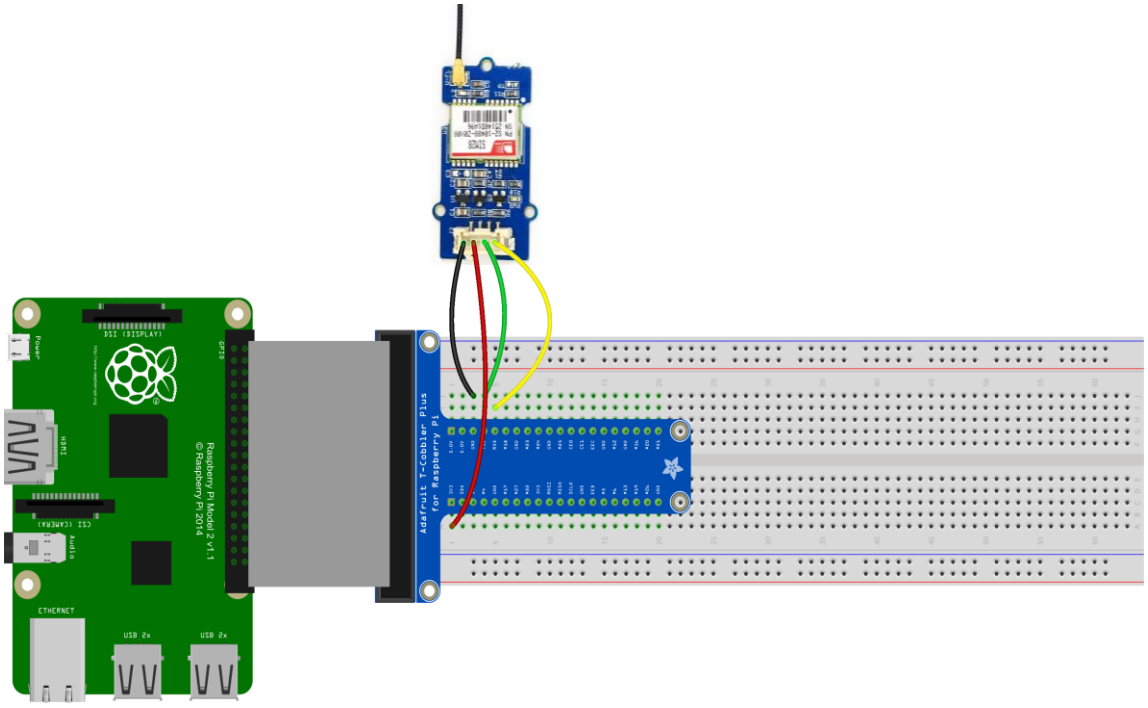


3 – UTILISATION DU MODULE GPS SEN10752P AVEC LA CARTE RASPBERRY

3.1 – Schéma de câblage

Le shield GPS est alimenté sous 3,3 V. La carte Raspberry ne va que recevoir des données issues du shield GPS. La connexion entre la broche Rx du shield et la broche GPIO15 (TXD) n'est pas nécessaire.





3.2 – Utilisation de l'UART

Pour établir la communication entre le shield GSM et la carte Raspberry Pi, il faut utiliser l'**UART matériel** du Raspberry Pi (HW UART). Il faut cependant libérer l'UART qui est réquisitionné pour l'utilisation du terminal. Pour libérer l'UART il faut réaliser les étapes suivantes :

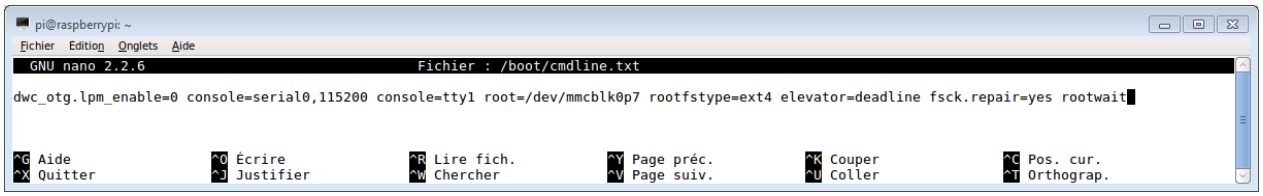
Modification du fichier « cmdline.txt »

Dans une fenêtre « Terminal », taper la ligne suivante :

```
sudo nano /boot/cmdline.txt
```

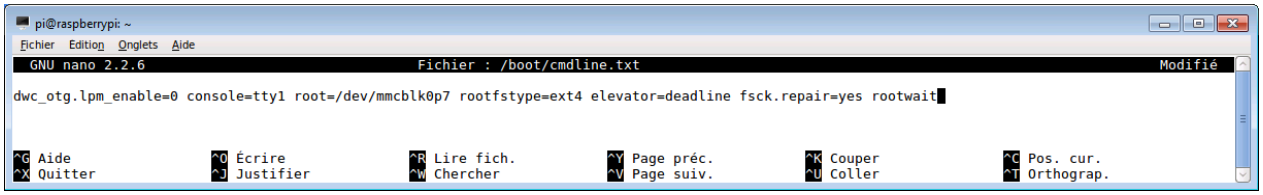
Modifier la ligne suivante :

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```



en :

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline rootwait
```



Modification du fichier « inittab »

Dans une fenêtre « Terminal », taper la ligne suivante :

```
sudo nano /etc/inittab
```

Placer la ligne suivante (si elle existe) en commentaire en plaçant le symbole # devant la ligne.

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Redémarrer le Raspberry Pi

Dans une fenêtre « Terminal », taper la ligne suivante :

```
sudo reboot
```

3.3 – Configuration de la liaison série logicielle

La carte Raspberry PI possède une **interface de communication série** UART accessible, grâce aux **broches GPIO14** (**TXD**) et **GPIO15** (**RXD**). La bibliothèque « **serial** » permet de gérer la communication série.

Importer la bibliothèque « Serial »

Pour importer la librairie « serial » dans un programme, on ajoutera au début du programme la ligne suivante :

```
import serial
```

Sous Raspbian, l'UART est nommée sous le nom de « **ttyAMA0** ». Toutes les données entrantes ou sortantes sont accessibles à partir du répertoire « **/dev/ttyAMA0** ».

Initialiser la liaison série

```
Nom_Liaison = serial.Serial ("/dev/ttyAMA0", baudrate=vitesse)
```

```
GPS = serial.Serial("/dev/ttyAMA0", baudrate=9600)
```

3.4 – Programme

Le programme suivant permet de lire l'ensemble des trames transmises par le shield GPS et de les afficher sur le moniteur série.

Programme Raspberry

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import serial, time, sys

GPS = serial.Serial('/dev/ttyAMA0',  9600, timeout = 0)   # Ouverture de la liaison série à 9600 bits/s
GPS.flush()          # Reset de la file d'attente liaison série

while True:
    Data=GPS.readline()      # Lecture des données récues sur la liaison série
    time.sleep(0.5)          # Pause 0,5
    print Data               # Affichage des données recues
```

Résultats terminal

```
*Python 2.7.9 Shell*
File Edit Shell Debug Options Windows Help
$GPGSV,3,2,12,33,36,200,39,14,24,315,23,19,18,040,14,29,17,192,41*71
$GPGSV,3,3,12,06,14,069,14,15,13,173,24,02,13,112,23,10,05,256,31*72
$GPRMC,093228.000,A,4449.7185,N,00033.9623,W,0.00,70.31,130617,,,D*44
$GPGGA,093229.000,4449.7185,N,00033.9623,W,2,10,0.78,20.3,M,49.7,M,0000,0000*43
$GPGSA,A,3,29,25,12,32,14,15,10,24,19,06,,,1.08,0.78,0.74*01
$GPGSV,3,1,12,12,73,032,30,24,57,112,27,25,56,264,42,32,40,298,30*79
$GPGSV,3,2,12,33,36,200,39,14,24,315,23,19,18,040,13,29,17,192,41*76
Ln: 6 Col: 0
```

<u>Coordonnées</u> :

Latitude : **4449.7185,N = 44,8286° Nord = 44° 49,7185′ Nord = 44° 49′ 43,11 ″ Nord**,

Longitude **00033.9623,W = 0,566° Ouest = 0° 33,9623′ Ouest = 0° 33′ 57,738 ″ Ouest**

Vérification du résultat sur « Google Maps »