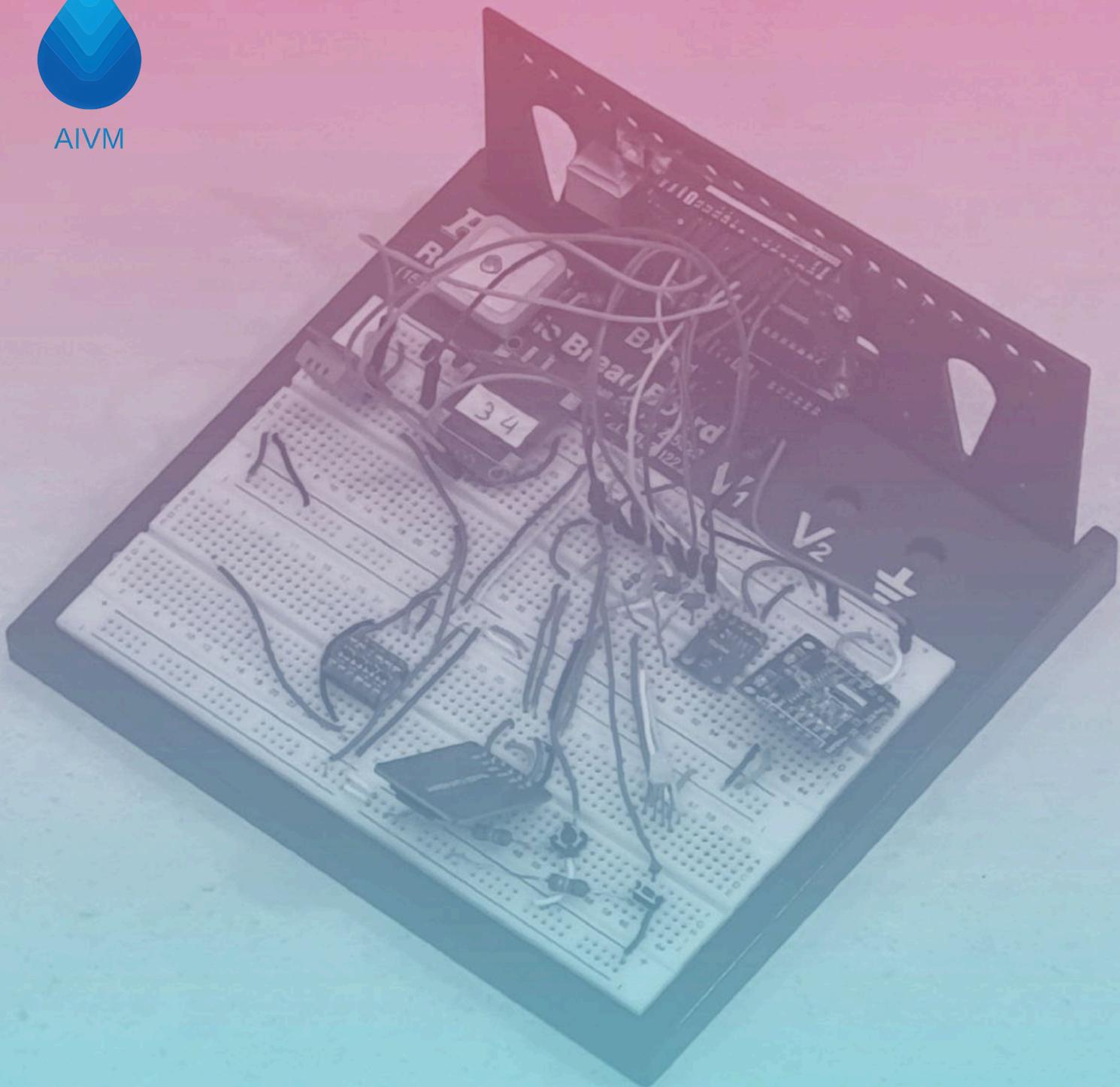




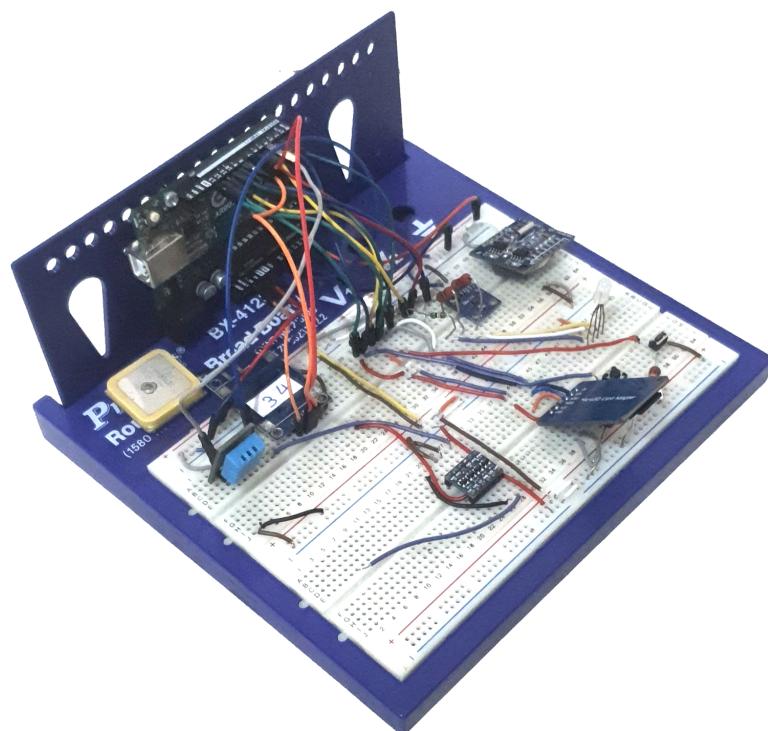
AIVM



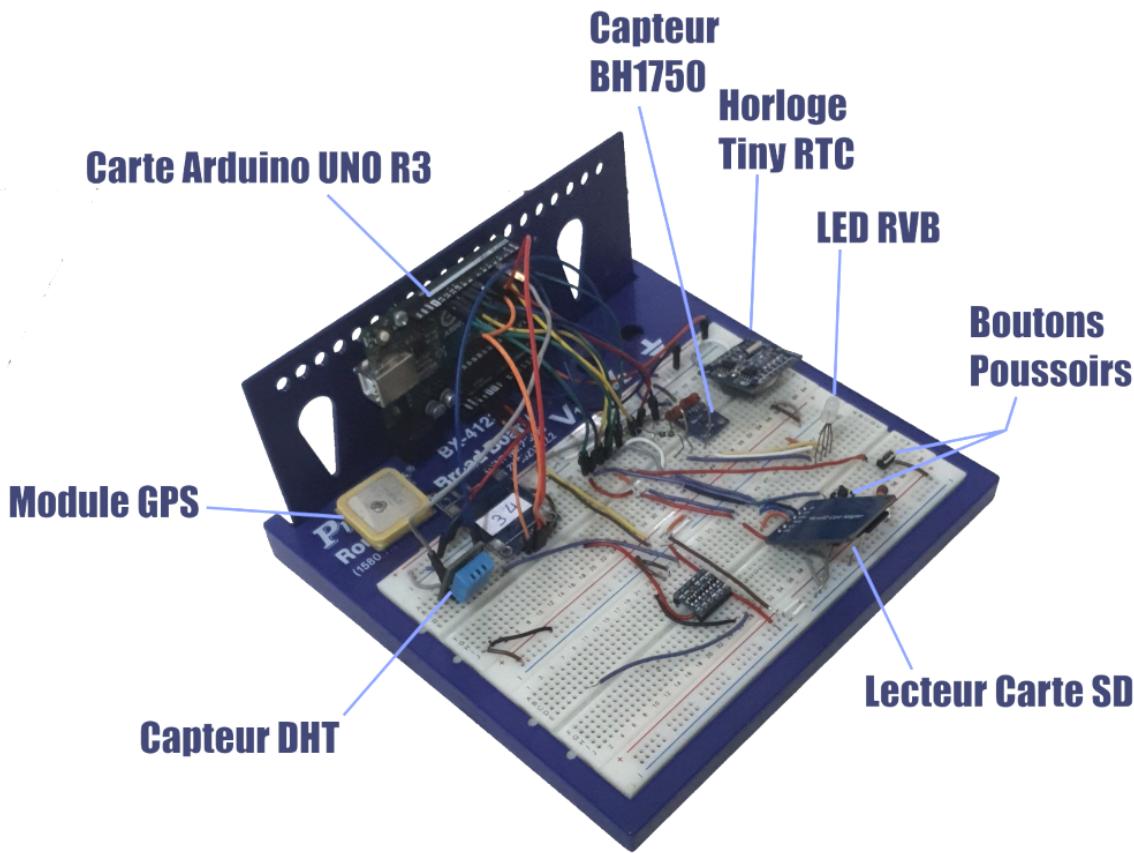
# Guide Technique – Station Météo Embarquée (AIVM<sup>1</sup>)

# 1-FONCTIONNEMENT GLOBALE DU SYSTÈME

Ce système est un dispositif de station météo pouvant être installé sur des navires. Il est composé de plusieurs composants, dont un microcontrôleur (ATmega328P sur Arduino Uno) qui gère toutes les opérations et traitements. Il interprète les entrées de l'utilisateur, telles que les appuis sur les boutons ou les commandes envoyées, lit les données des capteurs (DHT pour la température et l'humidité, BH1750 pour la luminosité), ainsi que celles du GPS et de l'horloge RTC, et les sauvegarde sur une carte SD. Le programme du microcontrôleur contient également une partie dédiée à l'affichage des erreurs, comme le dysfonctionnement d'un capteur ou une carte SD pleine.



## 2-SCHÉMA DU MONTAGE ET COMPOSANTS



### Carte Arduino UNO R3 (Microcontrôleur ATmega328P)

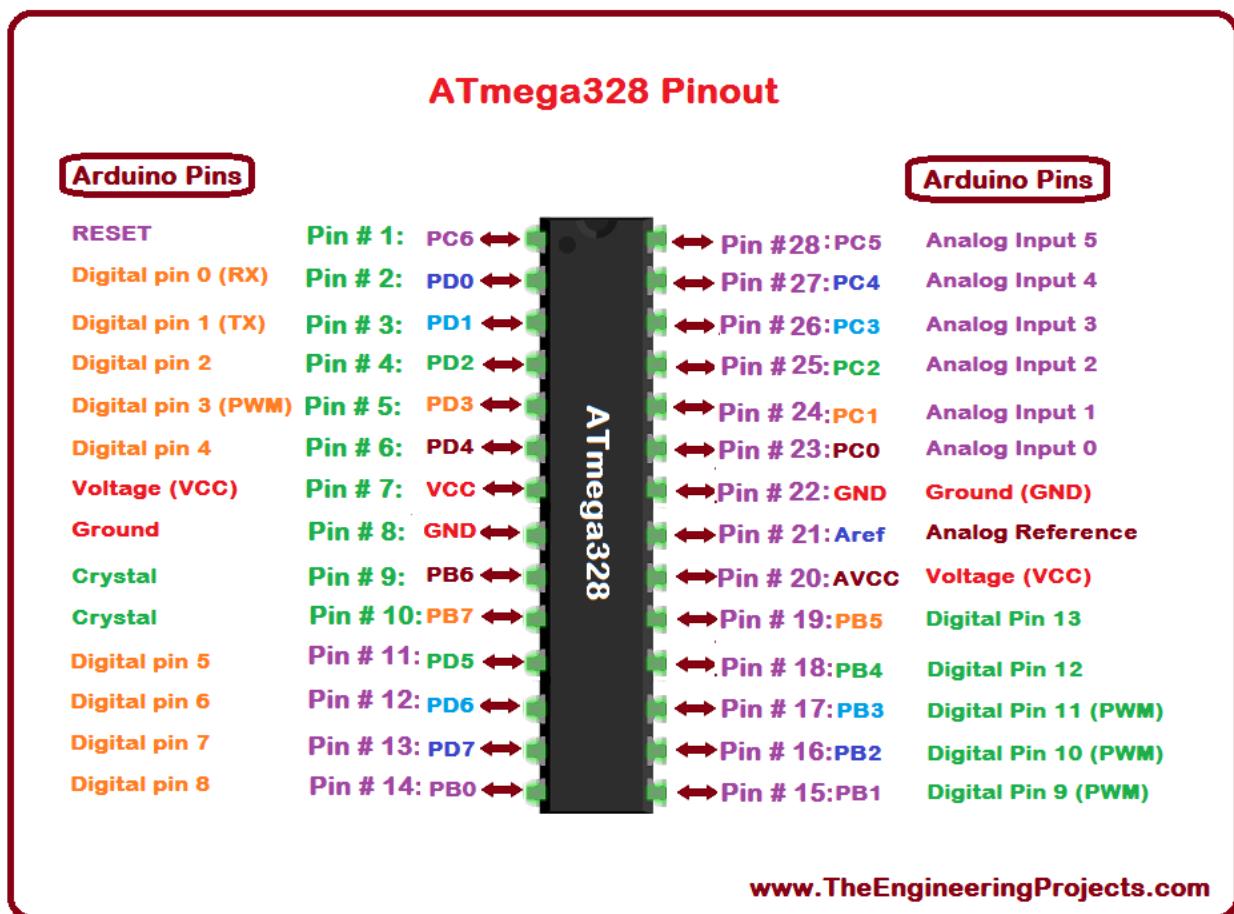
La carte Arduino UNO R3, basée sur le microcontrôleur ATmega328P, constitue le cœur du système. Fonctionnant à 16 MHz, elle dispose de 32 Ko de mémoire Flash, 2 Ko de SRAM et 1 Ko d'EEPROM.

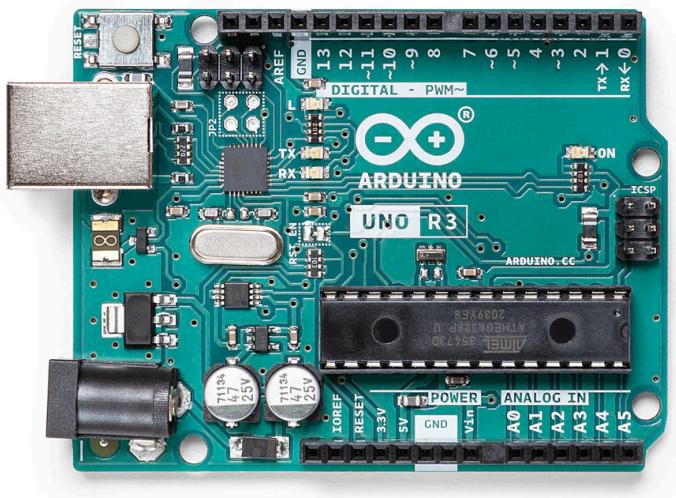
Elle gère la lecture des capteurs, la communication série avec les modules GPS et RTC, la sauvegarde des données sur carte SD et la détection des appuis sur les boutons.

Les interfaces UART, I<sup>2</sup>C et SPI sont exploitées simultanément :

- UART matériel (pins 0 et 1) réservé à la communication série pour les commandes et l'affichage,
- I<sup>2</sup>C (A4 = SDA, A5 = SCL) pour le capteur BH1750 et le module RTC,
- SPI (10 à 13) pour le lecteur de carte SD.

Les broches 2 et 3 sont utilisées pour les interruptions externes associées aux boutons poussoirs, assurant une réactivité immédiate aux appuis.





## Boutons Poussoirs

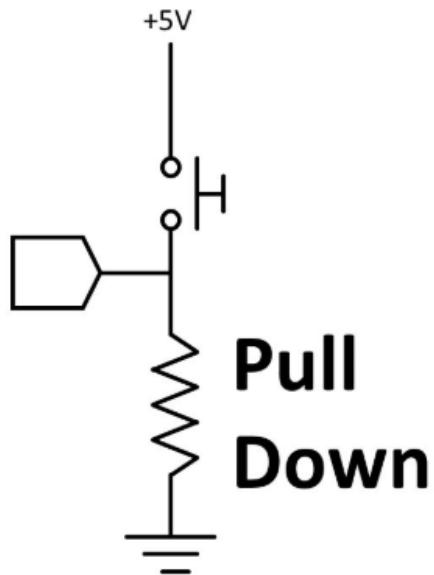
Deux boutons poussoirs sont intégrés pour naviguer entre les différents modes :

- Bouton rouge connecté à la broche 2 (INT0),
  - Bouton vert connecté à la broche 3 (INT1).



Les deux utilisent des résistances de pull-down externes de  $10\text{ k}\Omega$  afin de garantir un état bas stable au repos.

Chaque appui provoque une interruption externe, permettant au microcontrôleur de détecter instantanément l'action sans



interrompre les autres tâches.

Un appui long (~5 s) déclenche un changement de mode, tandis qu'un appui court exécute une commande ponctuelle.

La durée d'appui est mesurée à l'aide de la fonction millis().

### Capteur DHT (Température et Humidité)

Le capteur **DHT11** (ou DHT22 selon la version) est un capteur numérique combiné qui mesure la **température** et l'**humidité relative** de l'air.

Il est connecté à la **broche numérique 4** de l'Arduino et utilise une **résistance de tirage de 10 kΩ** reliée au +5 V pour stabiliser la ligne de données.



## Spécifications principales :

- **Alimentation** : 3,3 V à 5 V
- **Plage de température** : 0 à 50 °C ( $\pm 2$  °C de précision)
- **Plage d'humidité** : 20 à 90 % HR ( $\pm 5$  %)
- **Fréquence d'échantillonnage** : 1 mesure par seconde
- **Interface** : signal numérique propriétaire (1 fil)
- **Bibliothèque utilisée** : *DHTNew*, optimisée pour sa légèreté en mémoire et sa stabilité.

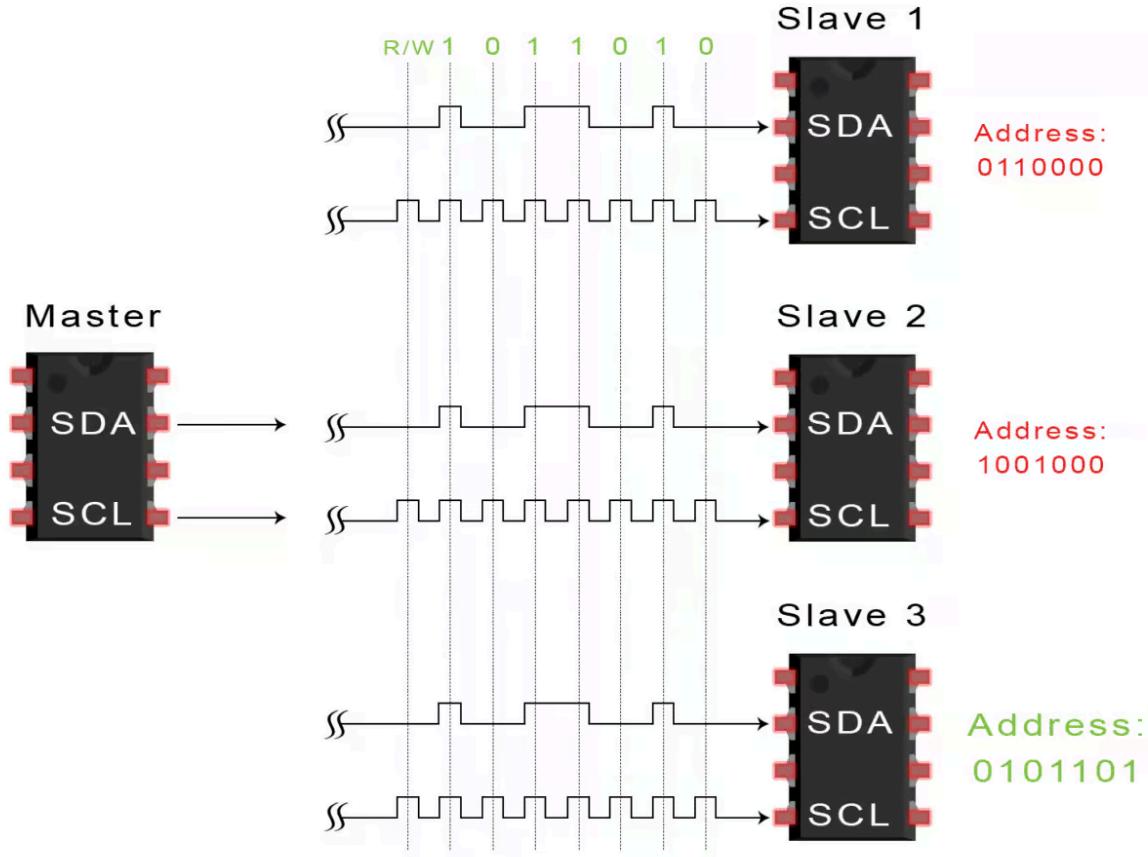
Ce capteur transmet les valeurs ambiantes à l'Arduino, qui les intègre ensuite aux enregistrements périodiques sauvegardés sur la carte SD.

## Capteur de Luminosité BH1750

Le **BH1750** est un capteur de luminosité numérique fonctionnant via le bus **I<sup>2</sup>C** partagé (SDA = A4, SCL = A5).

Il mesure l'intensité lumineuse ambiante en **lux**, offrant une grande sensibilité et linéarité sans besoin d'étalonnage manuel.





## Spécifications principales :

- **Alimentation : 3,3 V à 5 V**
- **Plage de mesure : 1 à 65 535 lux**
- **Résolution : 1 lux (mode haute précision)**
- **Interface : I<sup>2</sup>C (adresse par défaut 0x23)**

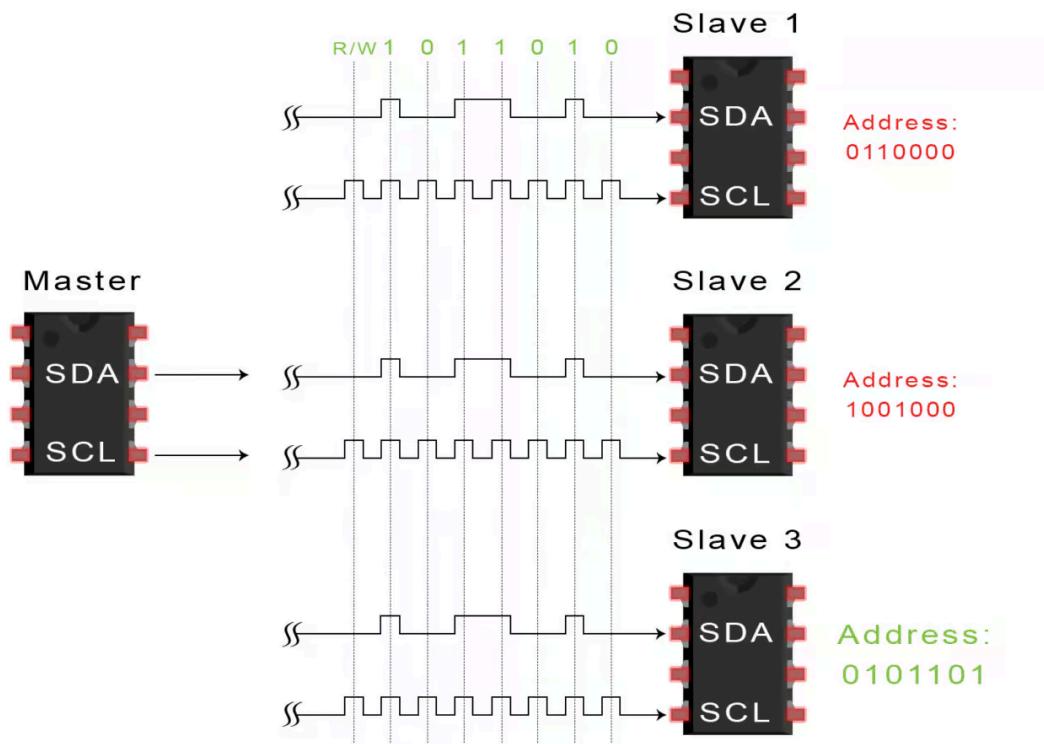
- **Temps de conversion** : environ 120 ms
- **Bibliothèque utilisée** : *BH1750* de claws.

Ce capteur fournit des mesures précises et rapides, idéales pour surveiller l'exposition lumineuse dans des environnements marins ou mobiles.

### Horloge Temps Réel DS1307 (Tiny RTC)

Le module **Tiny RTC** est basé sur le circuit **DS1307**, une horloge temps réel (RTC) fonctionnant via le bus **I<sup>2</sup>C**.

Il assure le maintien de l'heure et de la date, même lorsque le système est hors tension, grâce à sa **pile de sauvegarde CR2032**.





## Spécifications principales :

- **Alimentation** : 5 V
- **Interface** : I<sup>2</sup>C (SDA = A4, SCL = A5)
- **Précision** : ±2 min/mois
- **Mémoire additionnelle** : EEPROM 24C32 intégrée (32 Ko)
- **Horodatage** : format complet (année, mois, jour, heure, minute, seconde)

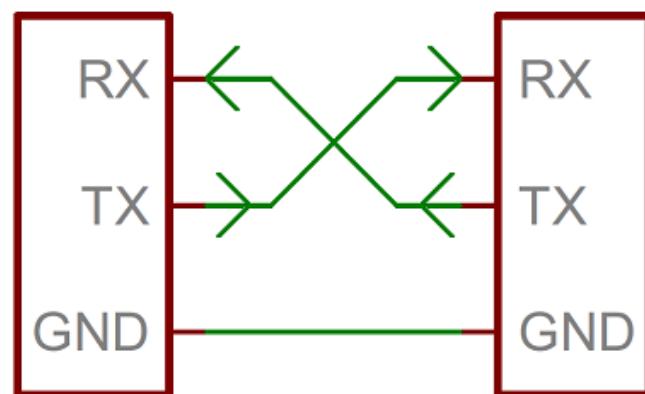
Les données temporelles fournies par le RTC permettent un **horodatage précis des mesures** dans les fichiers de journalisation.

## Module GPS (communication via NeoSWSerial)



Le module GPS utilisé (type NEO-6M ou équivalent) est connecté via une liaison **UART logicielle (NeoSWSerial)** utilisant les **broches 7 (RX)** et **8 (TX)** de l'Arduino.

Cette configuration libère le port série matériel (0-1), réservé à la communication de supervision et aux commandes UART.

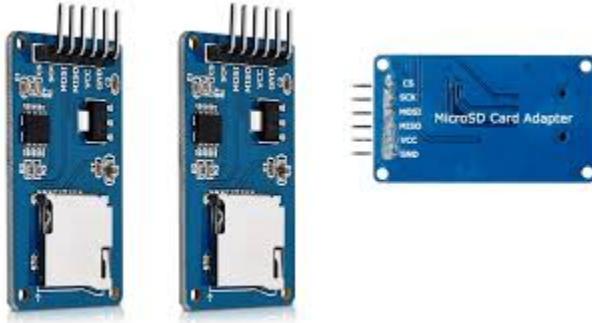


## **Spécifications principales :**

- **Alimentation** : 3,3 V à 5 V
- **Interface** : UART (9600 bps typique)
- **Protocoles** : NMEA 0183
- **Précision de position** :  $\pm 2,5$  m
- **Fréquence de mise à jour** : 1 Hz par défaut
- **Données fournies** : latitude, longitude

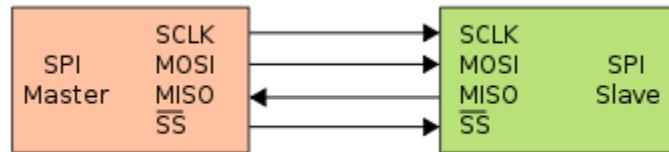
Les trames GPS sont traitées par le microcontrôleur pour synchroniser et enregistrer la position du navire.

## Lecteur de Carte SD



Le lecteur de carte SD communique avec le microcontrôleur via le **bus SPI** (CS=10, MOSI=11, MISO=12, SCK=13).

Il permet la **sauvegarde locale des mesures** sous forme de fichiers texte (.LOG ou .CSV) compatibles avec les tableurs.



## Spécifications principales :

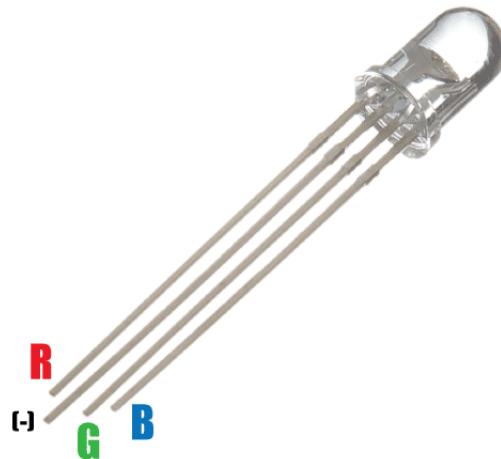
- **Tension de fonctionnement : 3,3 V à 5 V (avec régulateur intégré)**
- **Interface : SPI**
- **Capacité supportée : jusqu'à 32 Go (format FAT32)**

- **Bibliothèque utilisée : *SdFat*, avec configuration modifiée pour réduire son empreinte mémoire et activer les noms de fichiers longs (au-delà du format 8.3).**

Le système vérifie en permanence la disponibilité et la capacité de la carte SD.

En cas d'erreur (absence, pleine ou défectueuse), un code d'erreur est généré et affiché via la LED RVB et le port série.

### LED RVB



La **LED tricolore RVB** sert d'indicateur visuel du fonctionnement du système.

Elle est connectée aux **broches 5 (rouge)**, **6 (vert)** et **9 (bleu)** via des résistances de **220 Ω**.

Le contrôle se fait par **modulation de largeur d'impulsion (PWM)**, permettant de générer plusieurs couleurs selon les états du système.

#### Signification des couleurs :

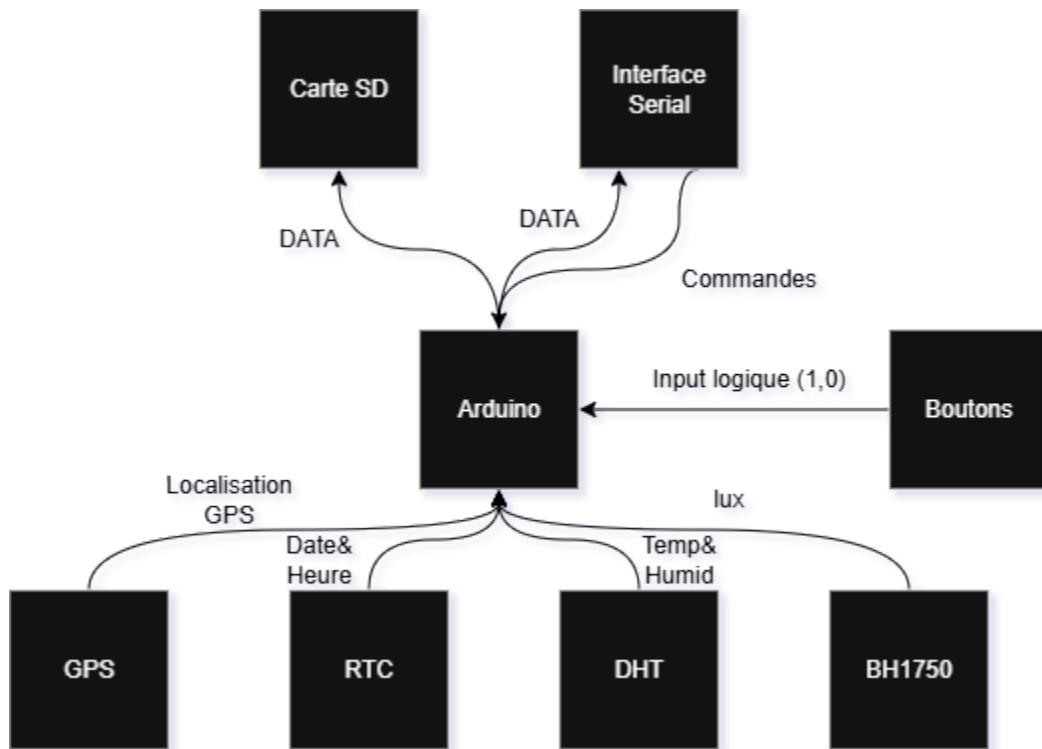
- **Vert** : mode standard (acquisition normale)

- **Bleu** : mode maintenance (accès série, stockage désactivé)
- **Rouge** : erreur (capteur, GPS ou carte SD)

## Spécifications principales :

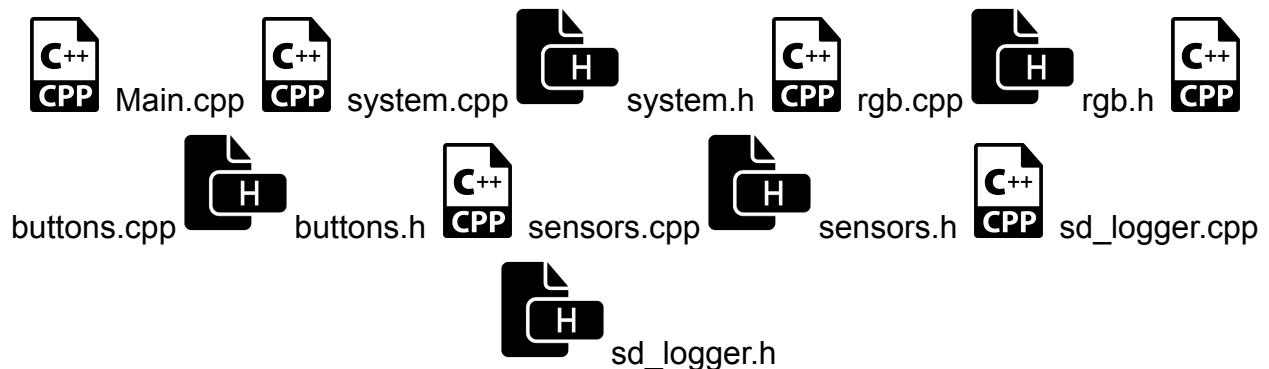
- **Alimentation** : 5 V
- **Type** : cathode commune
- **Courant maximal par canal** : 20 mA

## Schema du flux de données



## 3-ARCHITECTURE GENERALE PROGRAMME

Le programme est divisé en plusieurs fichiers .cpp et .h afin d'assurer une meilleure lisibilité, une organisation logique du code et de faciliter sa maintenance et son débogage.



 Main.cpp	Contient les fonctions de base <b>setup()</b> et <b>loop()</b> . Il s'occupe de la transition entre les différents modes, de l'initialisation du système au démarrage, ainsi que du système de commandes via UART.
 system.cpp	Contient des variables globales assignées, qui peuvent être modifiées pendant l'exécution du code et sont accessibles depuis l'ensemble du système.

 system.h	<p><b>Contient les déclarations de variables ainsi que les constantes <code>const</code> (stockées dans la mémoire flash).</b></p>
 rgb.cpp	<p><b>Contient le code qui contrôle la LED RGB avec différents presets de couleurs et de motifs, appelé lors des changements de mode ou en cas d'erreur.</b></p>
 rgb.h	<p><b>Initialisation RGB et accès aux valeurs globales via <code>#include &lt;system.h&gt;</code>.</b></p>
 buttons.cpp	<p><b>Contient les fonctions d'interruption associées aux deux boutons poussoirs, appelées lors de leurs changements d'état.</b></p>
 buttons.h	<p><b>Initialisation des fonctions définies dans <code>buttons.cpp</code> et accès aux valeurs globales via <code>#include &lt;system.h&gt;</code>.</b></p>
 sensors.cpp	<p><b>Contient le code qui assure la lecture des capteurs, de l'horloge Tiny RTC et du GPS. Il vérifie également à chaque lecture si les composants sont accessibles et, en cas de problème, renvoie le code d'erreur approprié.</b></p>
 sensors.h	<p><b>Initialisation des fonctions définies dans <code>sensors.cpp</code> et accès aux valeurs globales via <code>#include &lt;system.h&gt;</code>.</b></p>
 sd_logger.cpp	<p><b>Contient le code chargé de la sauvegarde des données sur la carte SD. Il vérifie également si la carte SD est accessible ou pleine et, en cas de problème, renvoie le code d'erreur approprié.</b></p>



sd\_logger.h

Initialisation des fonctions définies dans `sd_logger.cpp` et accès aux valeurs globales via `#include <system.h>`.

### Explication de systèmes clés :

-**L'acquisition des données** (capteurs, horloge RTC et GPS) s'effectue de manière cyclique. À la fin de chaque cycle, les données collectées, si elles sont complètes, peuvent être soit sauvegardées sur la carte SD, soit affichées via le port série UART.

Si, durant un cycle d'acquisition, l'accès aux données d'un composant échoue pendant toute la durée du **timeout** prédéfini, le système bascule en **mode erreur** : il interrompt les opérations normales et signale l'anomalie par un **code couleur spécifique** sur la LED RVB.

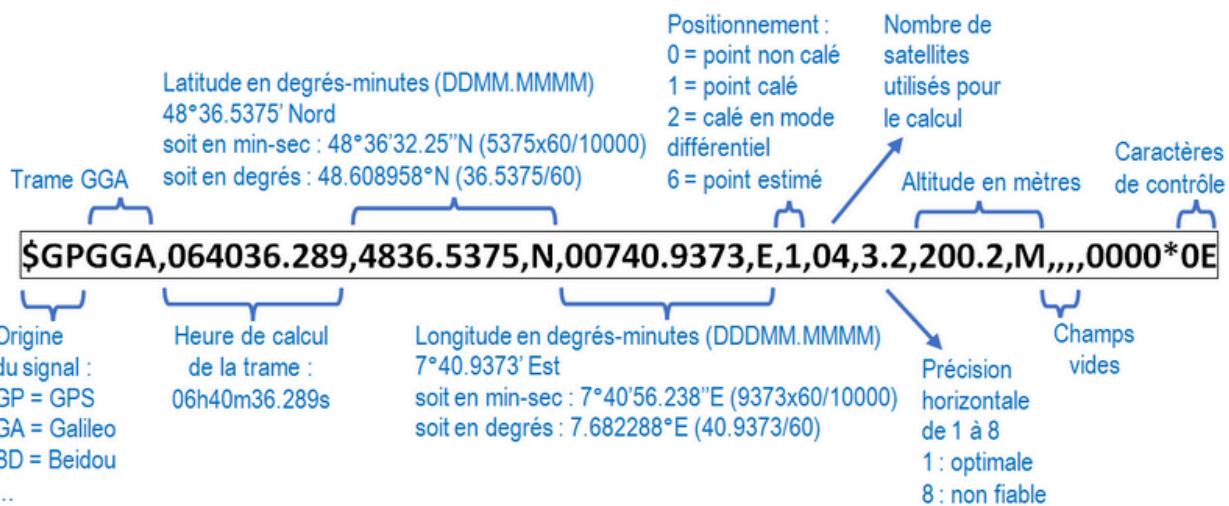
Une fois le cycle de clignotement terminé, le système vérifie de nouveau la disponibilité du composant défaillant. Si le problème persiste, il reste en mode erreur ; dans le cas contraire, il reprend automatiquement le **mode précédent**.

-**La fonction de gestion de la LED RVB est dynamique**, car elle intègre déjà l'ensemble des **prérglages de couleurs** correspondant aux différents cas d'utilisation (modes de fonctionnement et erreurs). Il suffit simplement de l'appeler avec le **numéro du prérglage** souhaité pour que la LED change progressivement de couleur par un **effet de dégradé fluide**.

```
void RGB_Control(uint8_t preset, uint8_t R, uint8_t G, uint8_t B)
```

-La fonction qui récupère les données GPS n'utilise aucune bibliothèque dédiée, afin d'économiser la mémoire Flash du microcontrôleur.

Le programme lit directement les **trames NMEA** envoyées par le module GPS via le port série logiciel (**NeoSWSerial**, sur les broches 7 et 8). Parmi ces trames, seules celles de type **\$GPGGA** ou **\$GPRMC** sont utilisées, car elles contiennent les informations de **latitude** et de **longitude**.



Le traitement consiste à analyser la chaîne reçue caractère par caractère, à identifier les séparateurs “,”, puis à extraire les champs correspondants à la latitude et à la longitude. Ces valeurs, initialement transmises sous forme de texte (degrés et minutes), sont ensuite converties en format décimal pour être plus facilement exploitables.

Les coordonnées obtenues sont ensuite stockées dans des variables globales, puis associées aux mesures des capteurs et à l'horodatage RTC lors de la sauvegarde sur la carte SD ou de l'affichage via le port série.

Cette méthode, entièrement codée manuellement, permet de réduire considérablement la taille du programme tout en conservant les informations géographiques essentielles.

## 4-COMPILATION ET TÉLÉVERSEMENT

Ce projet a été codé en utilisant l'extension **PlatformIO** sur **VS Code**. Elle inclut toutes les fonctionnalités nécessaires pour la compilation et le téléchargement, permettant d'automatiser ces processus. Nous avons conçu un script de compilation et de téléchargement qui utilise **PlatformIO**.

### ETAPE 1

Télécharger PlatformIO pour cela suivez cette vidéo youtube:

- ▶ Tutorial: Visual Studio Code install PlatformIO IDE extension

### ETAPE 2

Télécharger notre projet. **PlatformIO** contient la configuration, le code source et les bibliothèques nécessaires. Ensuite, ouvrir le projet dans **PlatformIO**.

[Projet\\_SE\\_GRP8.zip](#)

## ETAPE 3

Télécharger notre script de téléversement, qui utilise **PlatformIO** pour compiler et téléverser le programme sur la carte Arduino. Il suffit de spécifier au début du script le chemin vers le projet **PlatformIO**, puis de l'exécuter.

### Build\_Upload\_Groupe8.bat

```
@echo off
:: =====
:: PlatformIO Automation Script for Projet_SE_GRP8
:: =====
:: Make sure PlatformIO Core (CLI) is installed:
:: pip install platformio
:: And added to PATH

:: Go to your PlatformIO project folder
cd /d "C:\Users\NITRO\Documents\PlatformIO\Projects\Projet_SE_GRP8"
```

Si l'Arduino est connecté à un port COM, le script compilera ensuite le programme, le téléversera sur la carte, puis ouvrira l'interface série pour communiquer avec l'Arduino.

```
avrdude: AVR device initialized and ready to accept instructions
Reading | ##### | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file ".pio\build\uno\firmware.hex"
avrdude: writing flash (31668 bytes):

Writing | ##### | 100% 5.18s

avrdude: 31668 bytes of flash written
avrdude: verifying flash memory against .pio\build\uno\firmware.hex:
avrdude: load data flash data from input file .pio\build\uno\firmware.hex:
avrdude: input file .pio\build\uno\firmware.hex contains 31668 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 4.06s

avrdude: verifying ...
avrdude: 31668 bytes of flash verified

avrdude: safemode: Fuses OK (E:00, H:00, L:00)

avrdude done.  Thank you.

===== [SUCCESS] Took 13.54 seconds =====
```

```
=====
Opening serial monitor...
=====
--- Terminal on COM10 | 9600 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file, nocontrol, printable, send_on_enter, time
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Logged: 251101_2.log
Logged: 251101_2.log
Logged: 251101_2.log
Logged: 251101_2.log
```

## 5-MÉMOIRE UTILISÉE APRÈS COMPIRATION

```
RAM: [=====] 70.8% (used 1450 bytes from 2048 bytes)
Flash: [=====] 98.2% (used 31668 bytes from 32256 bytes)
```

Ce qui est dans les limites de l'acceptable, et la compilation est pré-optimisée pour réduire la taille du code grâce à build\_flags = -Os dans le fichier platformio.ini.

**FIN DU DOCUMENT**