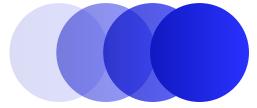


# Présentation de projet Mât Eolien 2025



# Sommaire

- Mes tâches à accomplir
- Présentation de la conception
- Présenter le planning prévisionnel et le planning réel
- Description des solutions et technologies utilisées
- Conclusion personnelle sur l'état d'avancement

# Mes tâches à accomplir



## ACQUERIR LES GRANDEURS PHYSIQUES ET TRANSMETTRE LES MESURES VIA UN BUS DE TERRAIN

- Analyser le cahier des charges et la structure globale du système.
- Rechercher les solutions existantes pouvant répondre au Cdc.
- Identifier les grandeurs caractéristiques des capteurs
- Programmer l'acquisition des conditions de vent et la transmission des mesures sur un bus CAN vers l'armoire centrale.
- Valider les mesures des conditions de vent
- Valider la communication des mesures à l'armoire centrale.
- Développer un module d'une application graphique (Qt) permettant au technicien de configurer les modules de mesures (numéro de mât, numéro de module etc...) et de diagnostiquer l'état de la chaîne de mesure.
- Intégrer cette partie au système global et valider le fonctionnement complet du système.

«requirement» MESURER LA VITESSE DU VENT
Id=1.1 Text= « La vitesse du vent doit être mesurée avec une précision de 3 km/h sur une gamme de 0 à 160km/h »

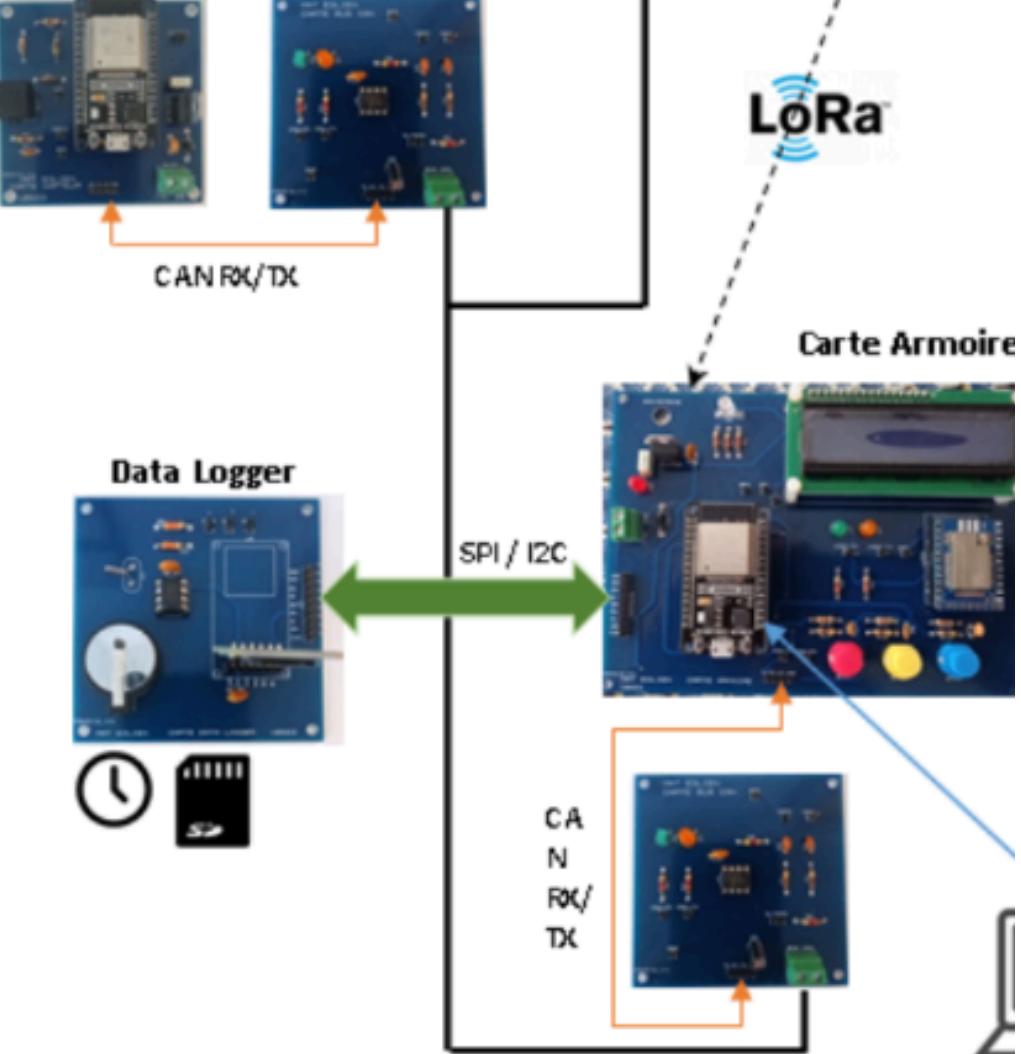
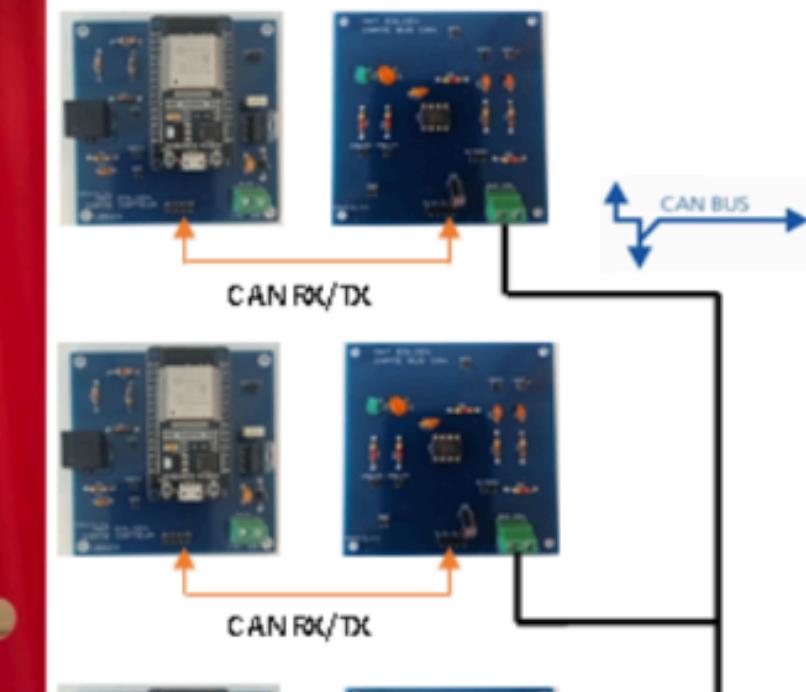
«requirement» MESURER LA DIRECTION DU VENT
Id=1.2 Text= « La direction du vent doit permettre de distinguer 16 secteurs de la rose des vents».

«requirement» COMMUNIQUER LES MESURES
Id=1.3 Text= « Les mesures à chaque hauteur doivent être communiquer vers l'armoire centrale située au pied du mât à une distance maximale de 200m »

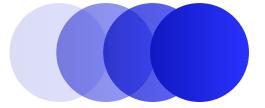
«requirement» BUS DE TERRAIN
Id=1.3.1 Text= « Les mesures sont transmises vers l'armoire via un réseau de terrain toute les 2 minutes »



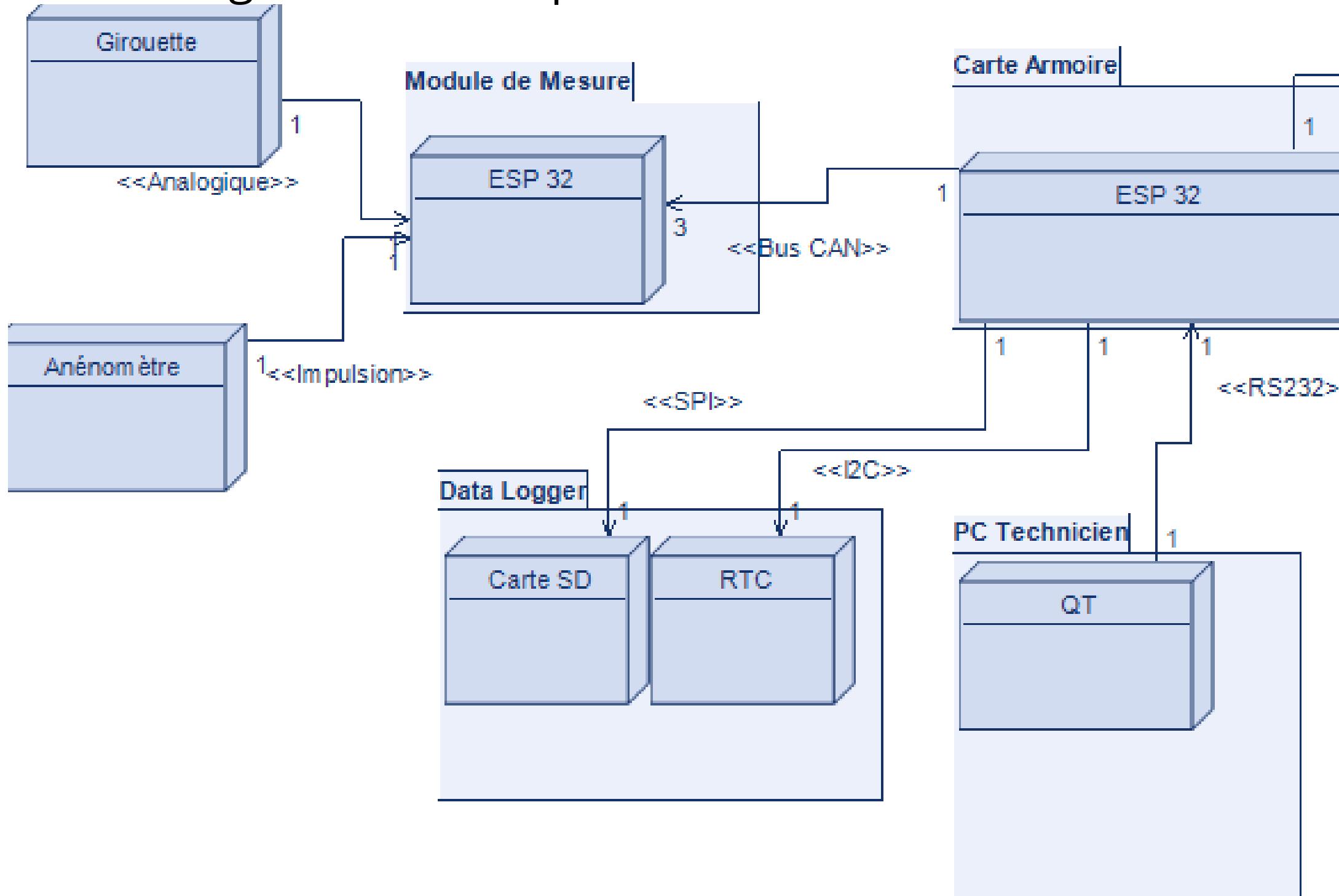
Carte capteur Interface bus CAN



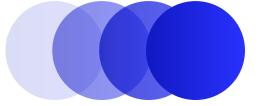
# Présentation de la conception



Extrait du diagramme de déploiement



# Présenter le planning prévisionnel et le planning réel



## Sprint 1

- [Elaborer le diagramme des classes \(Architecture\)](#) En cours [i](#)

Le diagramme des classes montrant les relations entre les classes du SPRINT1 est défini sous Modélio. Les attributs et méthodes d'une classe seront définis lors de la conception de la classe.... [+](#)

? · 7 mar 2025
- [Editer le diagramme avec Modélio](#)

[Editer le diagramme avec Modélio](#) Tristan Coquet
- [US : Acquérir les informations des capteurs](#) Terminé [i](#)

Budget 0 € Profil par défaut 12 h

[Acquérir les informations du capteur \(grouette\)](#) Tristan Coquet

[Acquérir les informations du capteur \(Anémomètre\)](#) Tristan Coquet

14 h · 7 mar 2025
- [Rechercher des programmes test](#) Terminé [i](#)

Budget 0 € Profil par défaut 14 h

[Rechercher les programmes du capteur anémomètres](#) Tristan Coquet

[Rechercher les programmes du capteur girouette](#) Tristan Coquet

[Comprendre les dispositions des pines sur ESP32](#) Tristan Coquet

[Comprendre la communication entre deux ESP32 WROOM](#) Tristan Coquet

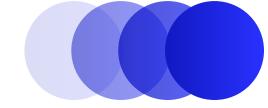
[Réaliser un programme test anémo](#) Tristan Coquet

[Réaliser un programme test girouette](#) Tristan Coquet

[Tester l'ajout du temps](#) Tristan Coquet

14 h · 7 mar 2025

# Présenter le planning prévisionnel et le planning réel



Sprint 1 Terminé i

Budget 0 € Profil par défaut 29 h

LISTE DES LIVRABLES + Ajouter

Elaborer le diagramme des classes (Architecture) Terminé i

Le diagramme des classes montrant les relations entre les classes du SPRINT1 est défini sous Modélio. Les attributs et méthodes d'une classe seront définis lors de la conception de la classe.

Éditer le diagramme avec Modélio Tristan Coquet

US : Acquérir les informations des capteurs Terminé i

Budget 0 € Profil par défaut 12 h

Acquérir les informations du capteur (girouette) Tristan Coquet

Acquérir les informations du capteur (Anémomètre) Tristan Coquet

Créer Objet Module Capteur Terminé i

Créer objet Girouette Tristan Coquet

Créer objet Anémomètre Tristan Coquet

Tester l'ajout des Classes Tristan Coquet

Faire un diagramme de classe Tristan Coquet

Commenter Classe Tristan Coquet

Optimiser Classe Tristan Coquet

Créer Objet Module Mesure Tristan Coquet

Mettre à jour le diagramme des classes (Architecture) En cours i

Mettre à jour le diagramme des classes (Architecture) » Mettre à jour le diagramme des classes (Architecture)

Le diagramme des classes est mis à jour (classes et relations). Les attributs et méthodes d'une classe seront définis lors de la conception de la classe.... i

Mettre à jour le diagramme des classes.

Mettre à jour le diagramme des classes. Tristan Coquet

«requirement»

**MESURER LA VITESSE DU VENT**

Id=1.1  
Text= « La vitesse du vent doit être mesurée avec une précision de 3 km/h sur une gamme de 0 à 160km/h »

«requirement»

**MESURER LA DIRECTION DU VENT**

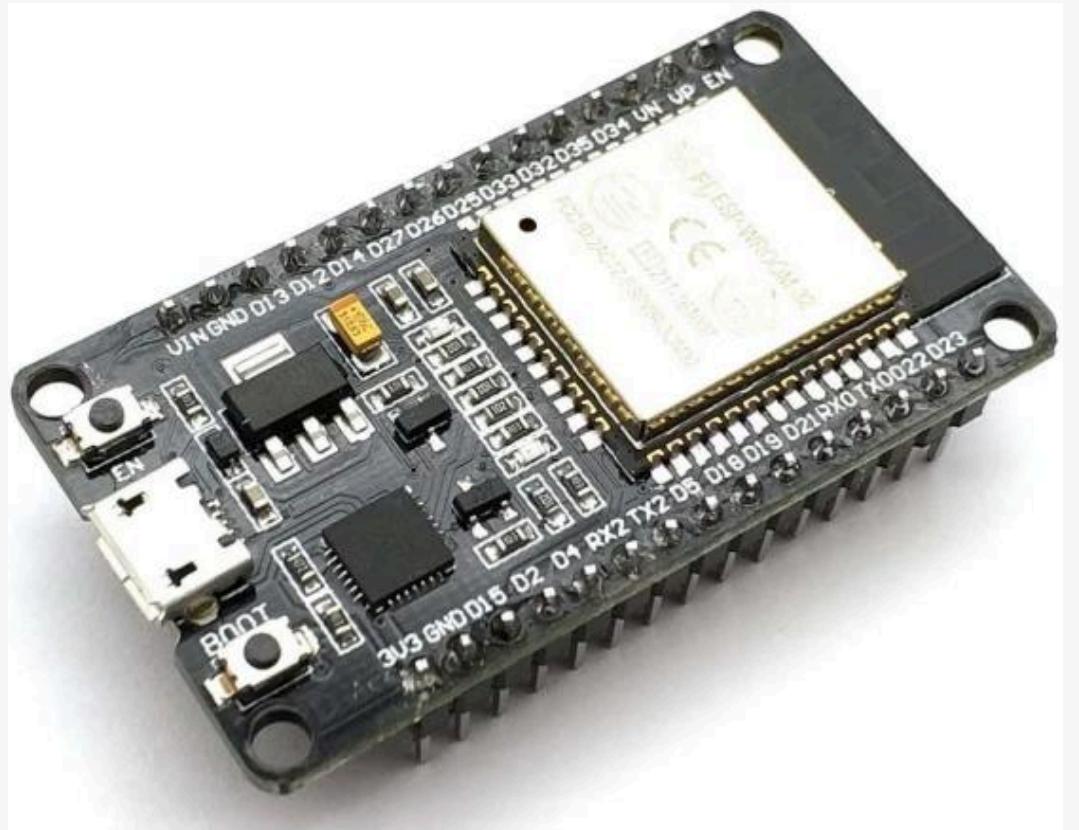
Id=1.2  
Text= « La direction du vent doit permettre de distinguer 16 secteurs de la rose des vents».

- Diagramme de classe
- Matérielle utilisé
- Imagination d'un test pour vérifier le programme de calcul de la vitesse (ID 1.1)
- Imagination d'un test pour vérifier le programme de détection de la direction du vent (ID 1.2)

# Description des solutions et technologies utilisées



Arduino IDE



Esp 32 Dev Module

Version Librairie choisit: 2.0.17  
(obsolète)

Dernière version disponible : 3.2

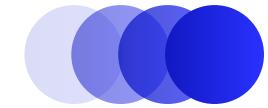


modèle : LEXCA002



modèle : LEXCA003

# Description des solutions et technologies utilisées



Information Cybersécurité



Esp 32 Dev Module

Version Librairie choisit: 2.0.17  
(obsolète)

Dernière version disponible : 3.2

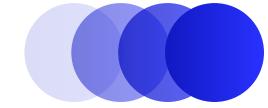
Version sujet à risque connue:

- CVE-2020-12638 (Bypass de l'authentification Wi-Fi)
- CVE-2020-13595 (Déni de service via Bluetooth Low Energy / BLE)
- CVE-2020-13594 (Crash du contrôleur BLE)
- CVE-2022-24893 (Corruption de mémoire dans le Bluetooth Mesh)
- CVE-2023-35818 (Contournement du Secure Boot et du chiffrement Flash via EMFI)

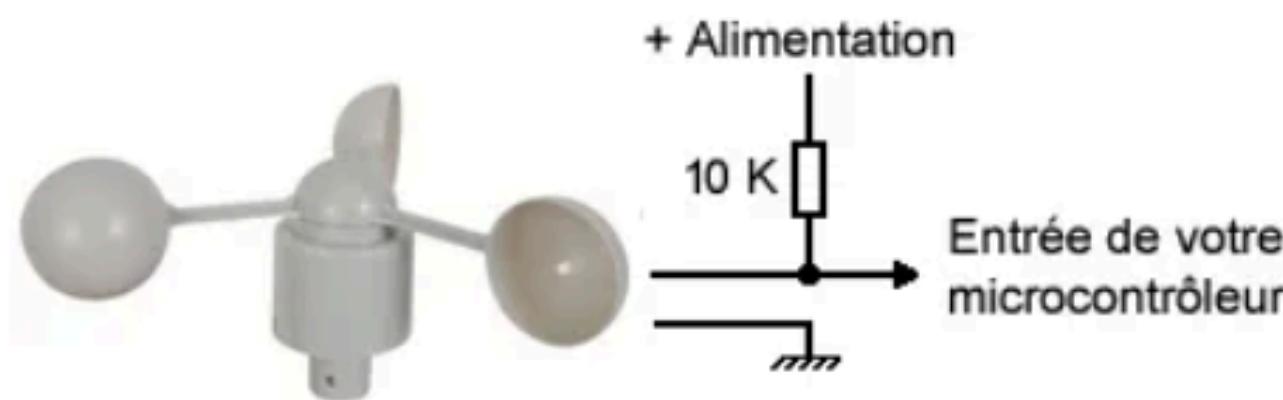
Version sujet à risque dans certain scénario :

- peu de compatibilité avec certaine librairies arduino
- **Version 3.X Sujette à problème avec les Esp Dev Module**
- Solution : la 2.0.17**

# Description des solutions et technologies utilisées



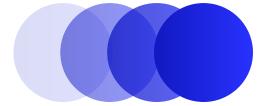
L'anémomètre intègre un petit aimant qui en passant devant un contact ILS ferme brièvement celui-ci lorsque les coupelles de l'anémomètre tournent (une vitesse de vent d'environ 2,4 km/h génère la fermeture impulsionnelle du contact par seconde). L'information du capteur est accessible via une prise RJ11 déportée sur un câble de 38 cm env.



modèle : LEXCA002

L'exploitation du capteur est très simple. Il vous suffira de lui associer une résistance de tirage au + de l'alimentation (3,3 à 5 Vcc) et de câbler les sorties du capteur entre cette résistance de tirage et la masse. Dès lors votre microcontrôleur devra simplement détecter les impulsions passagères (niveau logique bas) correspondant à la rotation d'un tour complet de l'anémomètre.

# Description des solutions et technologies utilisées



La girouette génère une variation de la valeur de sa résistance en fonction de son orientation. Cette variation n'est pas proportionnelle à la direction (la girouette délivre 16 valeurs de résistances différentes en fonction de sa position - cette dernière est donc uniquement capable de donner la direction du vent selon 16 positions). L'information du capteur est accessible via une prise RJ11 déportée sur un câble de 2.5 m.

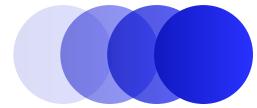
Vous trouverez la valeur de la résistance renvoyée par le capteur en fonction de l'angle de la girouette.

Angle	Résistance
0	33 Kohms
22.5	6.57 Kohms
45	8.2 Kohms
67.5	891 ohms
90	1 Kohms
112.5	688 ohms
135	2.2Kohms
157.5	1.41 K
180	3.9 Kohms
202.5	3.14K ohms
225	16 Kohms
247.5	14.12 Kohms
270	120 Kohms
292.5	42.12Kohms
315	64.9Kohms



modèle : LEXCA003

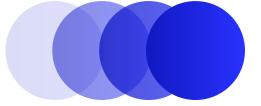
# Description des solutions et technologies utilisées



## Comparatif des différents BUS

Bus	Distance max fiable	Vitesse	Fiabilité	Maîtrise des collisions	Nombre de fils	Adapté à 200 m ?
UART (série)	~15 m (au mieux 50 m avec câble blindé)	Moyenne (jusqu'à 1 Mbps)	Faible (aucune correction d'erreur)	<b>Aucune gestion</b>	2 (TX/RX)	Non
I2C	~1-5 m max	Rapide (100 kHz-1 MHz)	Moyenne (pas prévu pour longues distances)	<b>Aucune gestion</b>	2 (+ GND)	Non
SPI	<1 m	Très rapide	Bonne à courte distance	<b>Aucun arbitrage</b>	4+ fils	Non
RS-485	Jusqu'à 1200 m	Moyenne (jusqu'à 10 Mbps)	Bonne (bus différentiel)	<b>! Simple collision detection !</b>	2 fils	Possible
CAN bus	Jusqu'à 1000 m (à 50 kbps)	Très bon (jusqu'à 1 Mbps à courte distance)	Très fiable (retransmission, CRC, détection d'erreur)	Gère les collisions (arbitrage par priorité)	2 fils	Solution Chosit

# Présenter le planning prévisionnel et le planning réel



## Sprint 2

### ☐ Crée Objet Module Capteur Terminé ⓘ

? · 28 mar 2025

- Créer objet Girouette Tristan Coquet
- Créer objet Anémomètre Tristan Coquet
- Tester l'ajout des Classes Tristan Coquet
- Faire un diagramme de classe Tristan Coquet
- Commenter Classe Tristan Coquet
- Optimiser Classe Tristan Coquet
- Crée Objet Module Mesure Tristan Coquet

### ☐ Mettre à jour le diagramme des classes (Architecture) En cours ⓘ

? · 28 mar 2025

Mettre à jour le diagramme des classes (Architecture) » Mettre à jour le diagramme des classes (Architecture)

Le diagramme des classes est mis à jour (classes et relations). Les attributs et méthodes d'une classe seront définis lors de la conception de la classe.... ⓘ

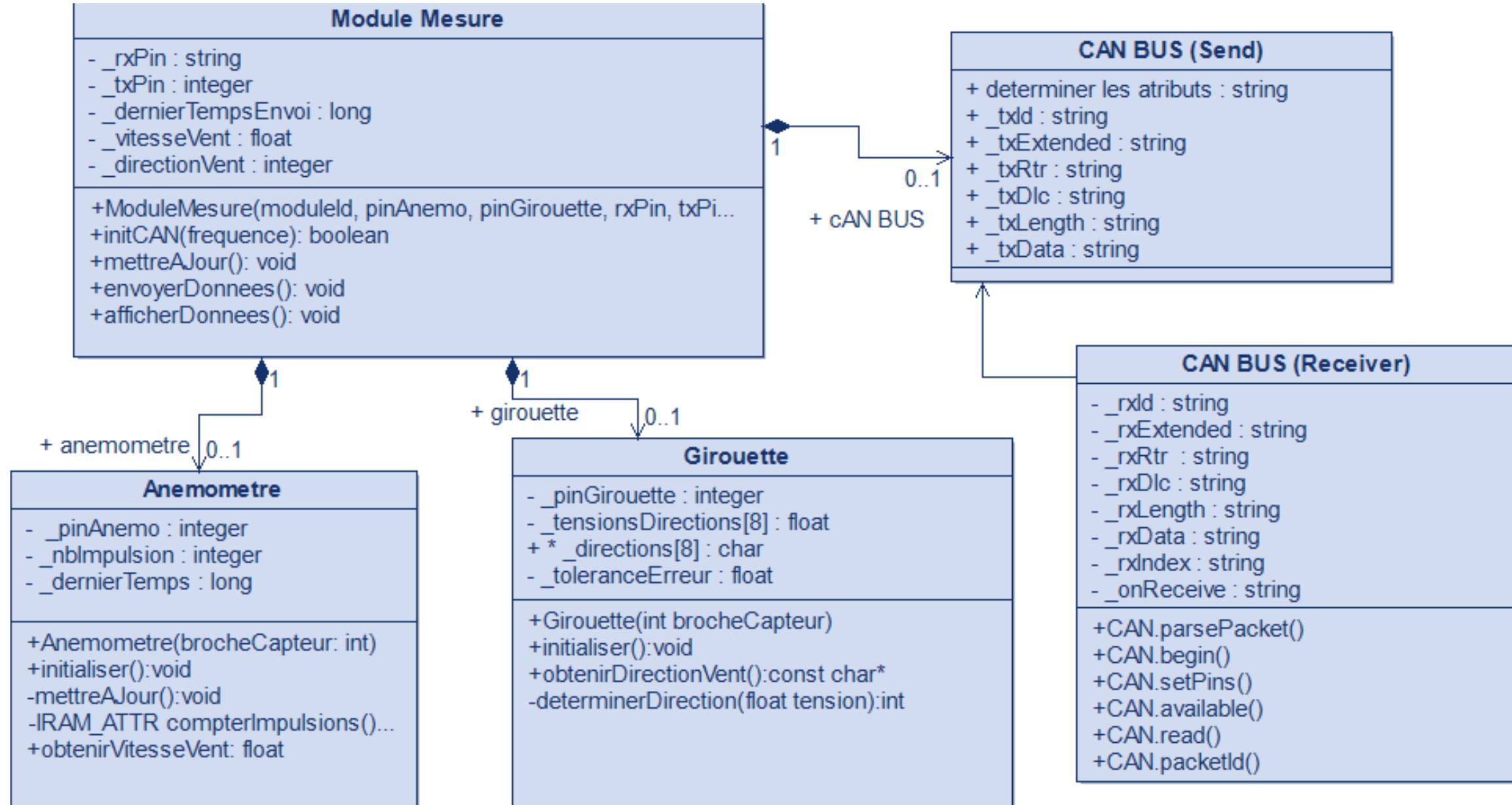
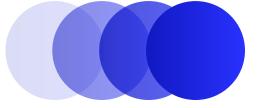
- Mettre à jour le diagramme des classes.
- Mettre à jour le diagramme des classes. Tristan Coquet

### ☐ US : Communiquer les informations des capteurs Terminé ⓘ

12 h · 28 mar 2025

Budget 0 € Profil par défaut 12 h

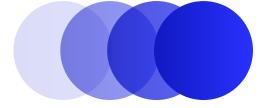
- Communiquer les mesures Tristan Coquet
- Fournir une fiche avec les mesures obtenue pour prouver le fonctionnement Tristan Coquet
- Mettre à jour le prg Tristan Coquet



Avantage d'avoir des classes

- Organisation du code → plus clair et modulaire
- Réutilisation
- Abstraction des détails non utiles

Diagramme de Classe



## Anemometre.h

```
#ifndef ANEMOMETRE_H
#define ANEMOMETRE_H

#include <Arduino.h>

// Fonction d'interruption globale pour compter les impulsions
void IRAM_ATTR compterImpulsions();

class Anemometre {
public:
    // Constructeur : initialise l'anémomètre avec sa broche
    Anemometre(int pinAnemo);

    // Initialise le capteur (configuration des broches, interruption, etc.)
    void initialiser();

    // Met à jour les valeurs du capteur (doit être appelé régulièrement)
    void mettreAJour();

    // Retourne la vitesse du vent en km/h
    int obtenirVitesseVent();

private:
    int _pinAnemo;      // Broche reliée à l'anémomètre
    volatile unsigned int _nbImpulsion; // Nombre d'impulsions comptées
    unsigned long _dernierTemps; // Dernier moment où l'anémomètre a été lu
};

#endif
```

## Grouette.h

```
#ifndef GIROUETTE_H
#define GIROUETTE_H

#include <Arduino.h>

class Grouette {
public:
    // Constructeur : initialise la girouette avec sa broche
    Grouette(int pinGrouette);

    // Initialise le capteur
    void initialiser();

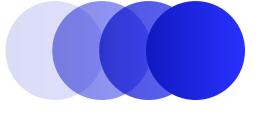
    // Retourne la direction du vent sous forme d'entier (1-8)
    const uint8_t obtenirDirectionVent();

private:
    int _pinGrouette; // Broche reliée à la girouette

    // Tableau des tensions de sortie de la girouette pour différentes directions du vent
    float _tensionsDirections[8] = {2.44, 1.36, 0.17, 0.46, 0.79, 1.89, 3.15, 2.81};
    //N:2.44 ,NE:1.36 ,E:0.17 ,SE:0.46 ,S:0.79 ,SO:1.89 ,O:3.15 ,NO:2.44
    // Tableau des directions du vent sous forme d'entiers (1-8)
    uint8_t _directions[8] = {1, 2, 3, 4, 5, 6, 7, 8}; // 1=Nord, 2=Nord-Est, etc.
    //uint8_t _directions[4] = {1, 2, 3, 4};
    float _toleranceErreur = 0.2; // Marge d'erreur tolérée pour comparer une tension

    // Trouve la direction du vent la plus proche en fonction de la tension lue
    int determinerDirection(float tension);
};

#endif
```



## ModuleMesure.h

```
#ifndef MODULE_MEASURE_H
#define MODULE_MEASURE_H

#include <Arduino.h>
#include <CAN.h>
#include "Anemometre.h"
#include "Girouette.h"

class ModuleMesure {
public:
    ModuleMesure(uint8_t moduleId, int pinAnemo, int pinGirouette, int rxPin, int txPin, uint32_t canId);

    bool initialiser();
    void mettreAJour();
    void afficherDonnees();
    void envoyerDonnees();

    void definirVitesse(uint8_t vitesse);
    void definirDirection(uint8_t direction);

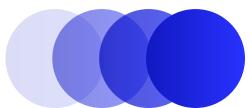
private:
    bool initCAN(long frequence = 500E3);

    uint8_t _moduleId;
    int _pinAnemo;
    int _pinGirouette;
    int _rxPin;
    int _txPin;
    uint32_t _canId;
    unsigned long _dernierTempsEnvoi;

    uint8_t _vitesseVent;
    uint8_t _directionVent;

    Anemometre _anemometre;
    Girouette _girouette;
};

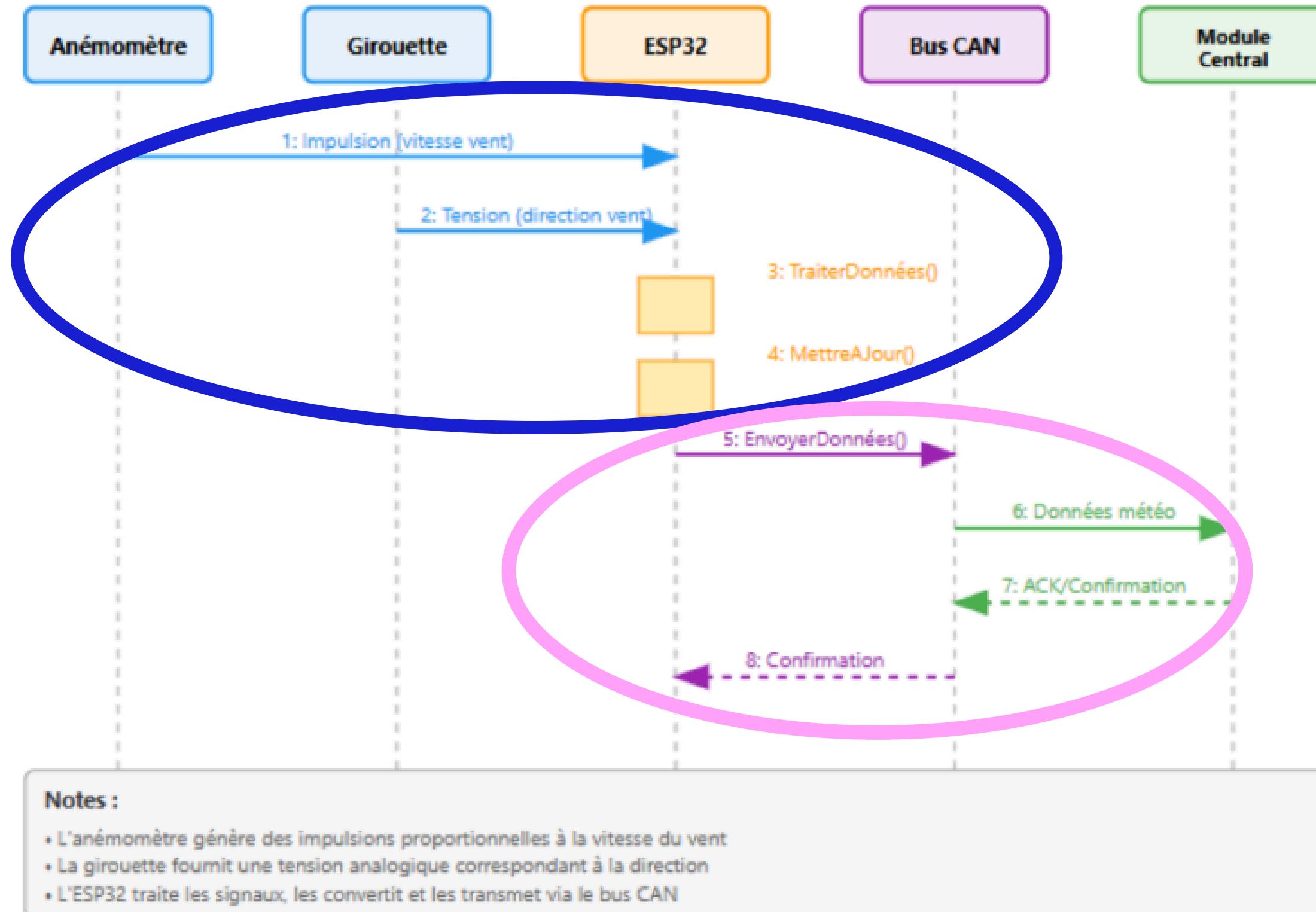
#endif // MODULE_MEASURE_H
```

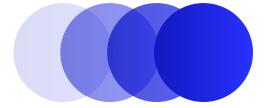


Plans de test unitaire  
(Anémomètre):

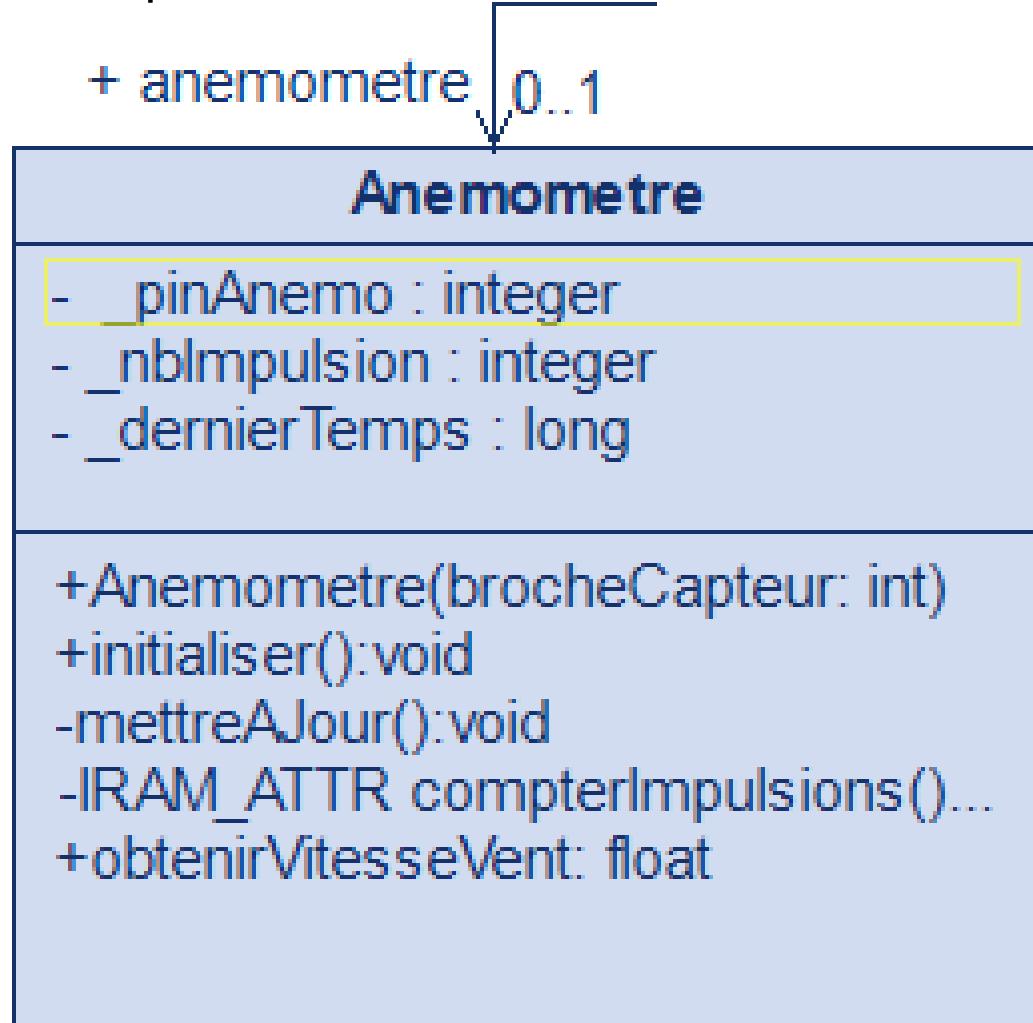
- Comparé vitesse réel et vitesse calculé au niveau du programme
- Comparé direction réel et la direction déterminer au niveau du programme
- Vérifier la communication CAN avec une trame de mesure

Diagramme de séquence : Acquisition et transmission des données





Test Unitaire et Intégration de la classe Anémomètres et calcul de vitesse pour confirmer bon fonctionnement



**Vitesse  
Anémomètre de  
Laboratoire**

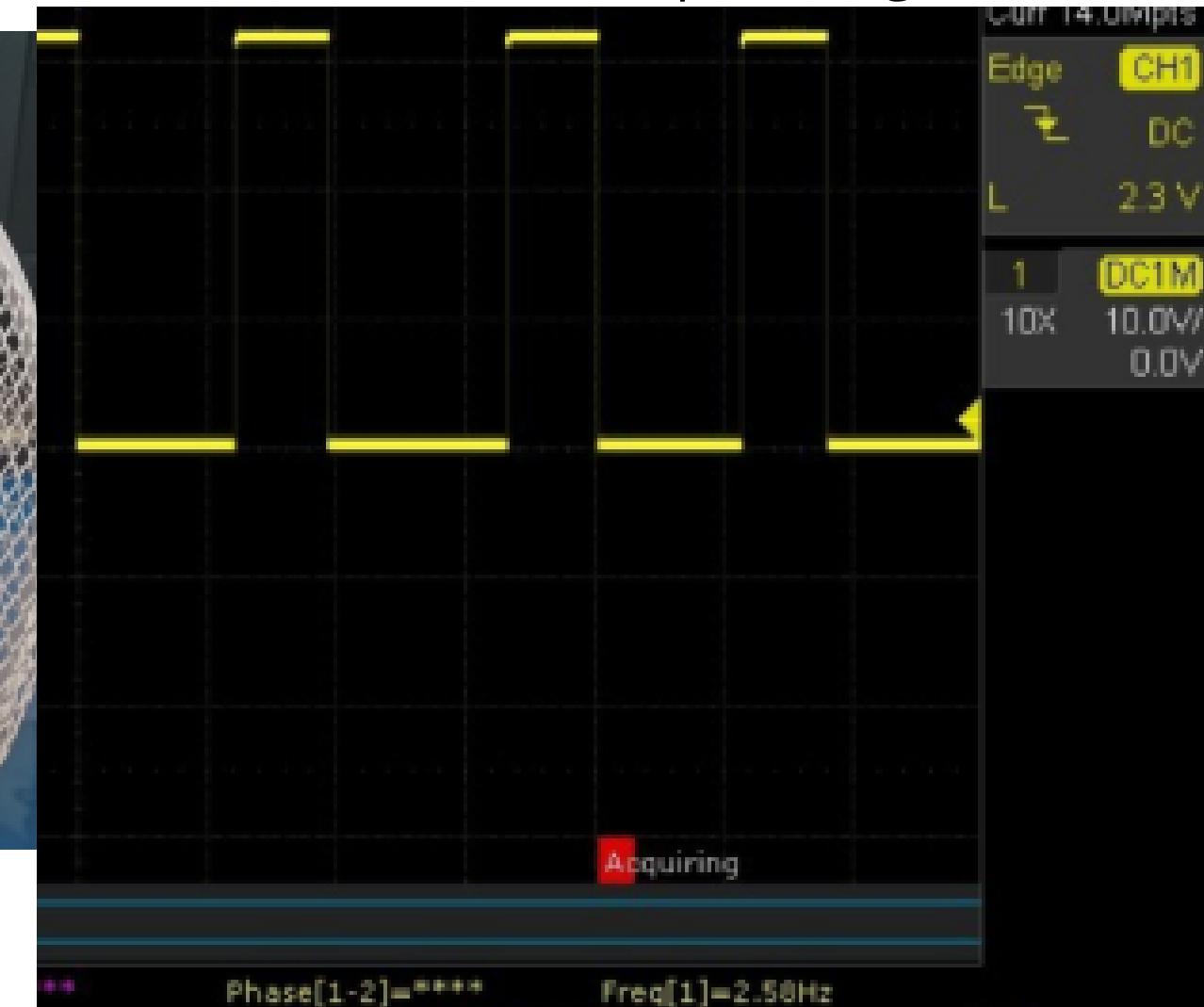
**(km/h)**

Exigence à respecter

~ 5,7



Relevé à l'oscilloscope du signal



```

-----
Module : 1
Vitesse du vent : 7 km/h
Direction du vent : 7
-----
```



Comptage Impulsion code:

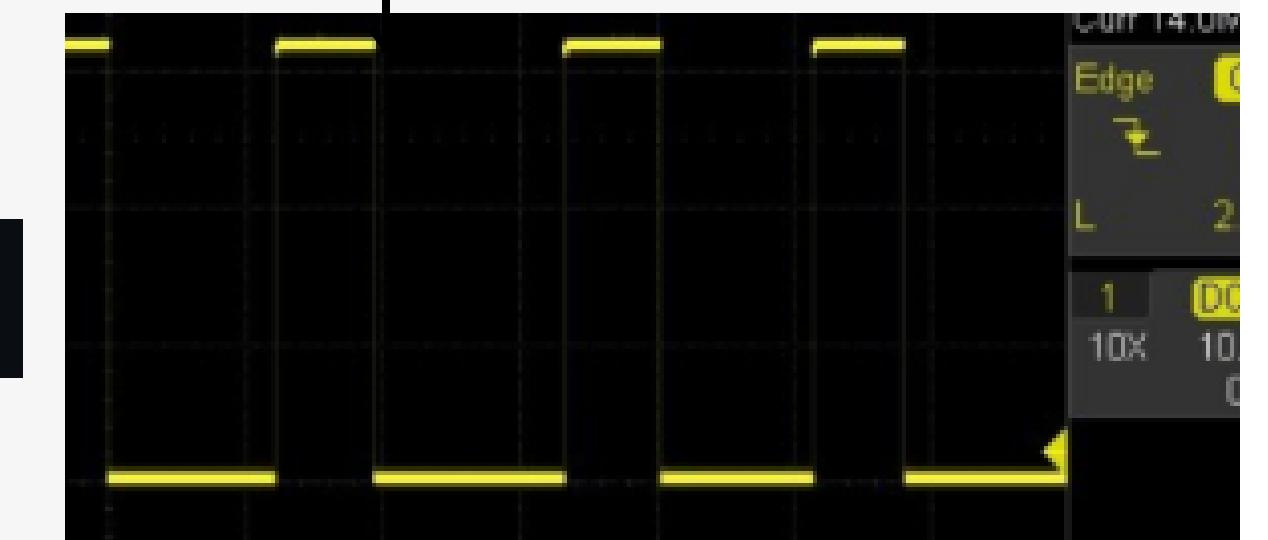
```
attachInterrupt(digitalPinToInterruption(_pinAnemo), compterImpulsions, FALLING);
```

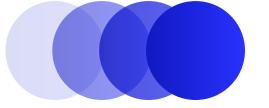
Attache une fonction à appeler quand un événement se produit

ISR (Interrupt Service Routine)  
Apelle de ...

```
void IRAM_ATTR compterImpulsions() {  
    impulsions++;
```

En cas de Front descendant (1 → 0)





## Résultat du test unitaire

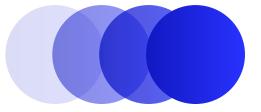
<i>Mode</i>	<i>Vitesse</i>	<i>Fréquenc</i>	<i>Vitesse</i>	<i>Vitesse</i>	<i>% d'erreur</i>	<i>Exigence</i>
<i>Vitesse du Ventilateur</i>	<i>Anémomètre de Laboratoire</i>	<i>Obtenue via Oscilloscope (Hz)</i>	<i>Calculer (fréquence *)</i>	<i>Calculer (Programme)</i>		<i>Id 1.1 Respecter</i>
1	~ 5,7	~ 2,58	~ 6,19	~ 7	13,08	Oui
2	6 ±	~ 3,16	~ 7,58	~ 9	18,72	Oui
3	~ 7,9	~ 3,53	~ 8,47	~ 9	6,26	Oui

## Test Concluant \*

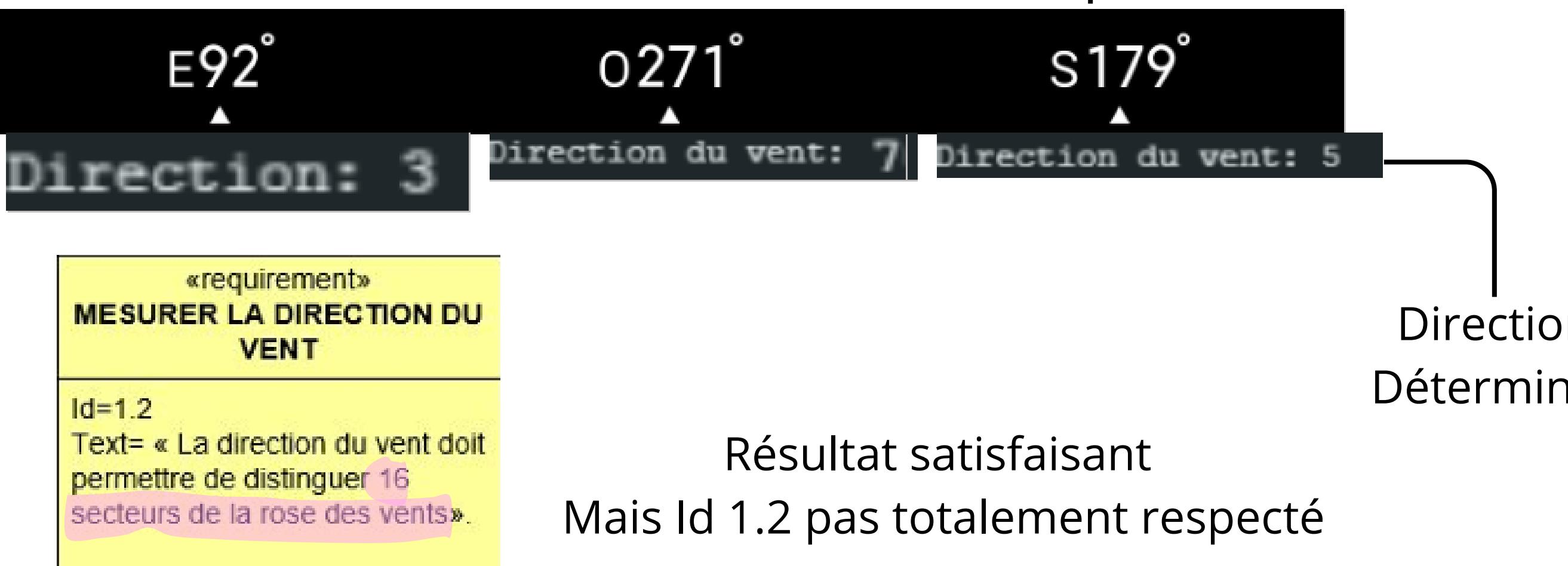
«requirement»  
**MESURER LA VITESSE DU VENT**

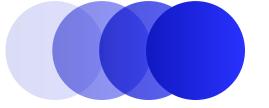
---

Id=1.1  
Text= « La vitesse du vent doit être mesurée avec une précision de 3 km/h sur une gamme de 0 à 160km/h »



Test Unitaire et Intégration de la classe Girouette et de détermination de la direction pour confirmer le bon fonctionnement





Détermination de direction code:

```
// Méthode qui retourne la direction du vent sous forme d'entier (1-8)
const uint8_t Girouette::obtenirDirectionVent() {
    int n_CAN = analogRead(_pinGirouette); //valeurAnalogique
    float tension = n_CAN * (3.3 / 4095);
    int indice = determinerDirection(tension); //uint8_t
    //Serial.println("Direction vent");
    return indice >= 0 ? indice + 1 : 0; // 1 à 8 (au lieu de _directions[indice])
    //Serial.println(tension);
}
```

- lecture de la valeur numérique de la girouette
- convertie cette valeur en tension
- Comparaison avec la liste des tensions pré-définies  
{2.44, 1.36, 0.17, 0.46, 0.79, 1.89, 3.15, 2.81}

N    NE    E    SE    S    SO    O    NO

03

# Présenter le planning prévisionnel et le planning réel

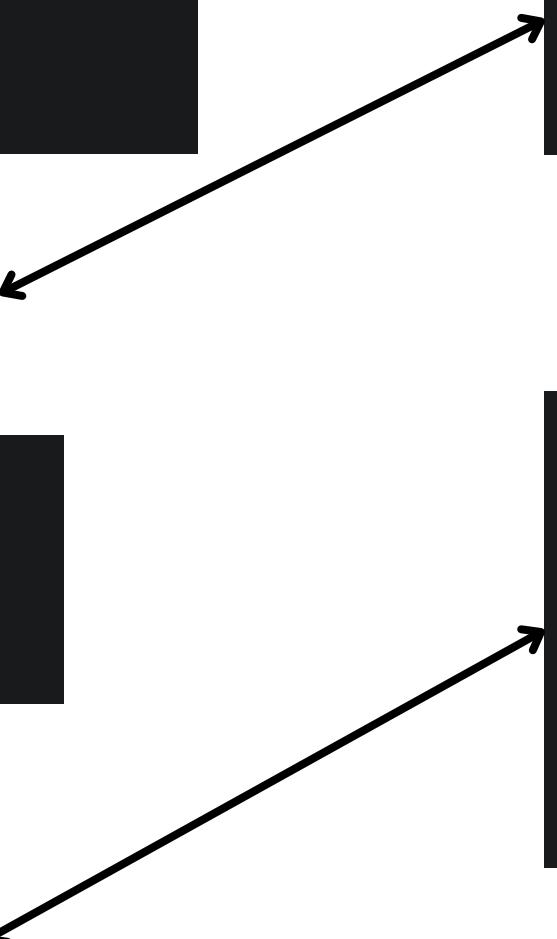


## Sprint 3

- TS: Simuler Module En cours i
  - | Budget 0 € Profil par défaut 6 h
  - Simuler Module 2 Tristan Coquet
  - Simuler Module 3 Tristan Coquet
  - Problèmes Tristan Coquet
  
- TS:Interface graphique QT En cours i
  - Développement app QT Tristan Coquet
  
- Tester la communication CAN Terminé i
  - Test communication module 1 vers Receveur Tristan Coquet
  - Test communication module 2 et 3 vers Receveur Tristan Coquet
  
- TS: Finalisation des différent programme Terminé i
  - Finalisation Programme Test Anémomètre Tristan Coquet
  - Finalisation Programme Test Girouette Tristan Coquet
  - Finalisation programme Test ModuleMesure Tristan Coquet
  - Finalisation programme Module 1 Tristan Coquet
  - Finalisation du programme module 2 et 3 Tristan Coquet
  - Finalisation du programme Receveur Tristan Coquet

## Sprint 4

- TS:Interface graphique QT En cours i
  - Développement app QT Tristan Coquet
  
- TS: Finalisation des différent programme Terminé i
  - Finalisation Programme Test Anémomètre Tristan Coquet
  - Finalisation Programme Test Girouette Tristan Coquet
  - Finalisation programme Test ModuleMesure Tristan Coquet
  - Finalisation programme Module 1 Tristan Coquet
  - Finalisation du programme module 2 et 3 Tristan Coquet
  - Finalisation du programme Receveur Tristan Coquet

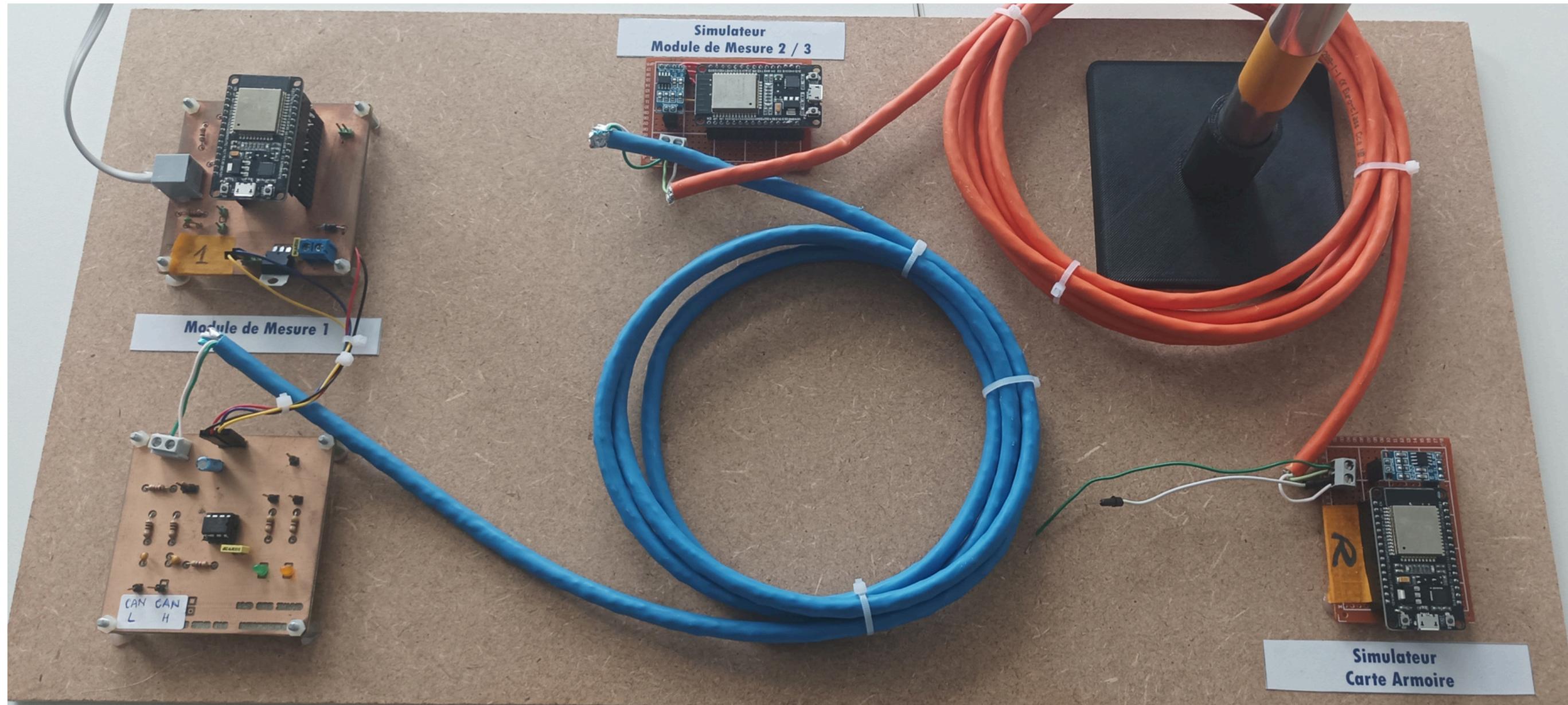


02

## Présentation de la conception



Test Unitaire CAN et vérifier le bon envoie d'un message



**CAN** by Sandeep Mistry ...  
<[sandeep.mistry@gmail.com](mailto:sandeep.mistry@gmail.com)>

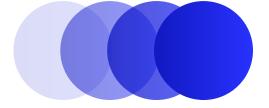
**0.3.1 installed**

An Arduino library for sending and receiving data using CAN bus.  
Supports Microchip MCP2515 base...

[More info](#)

**0.3.1** **REMOVE**

Envoyer une trame de mesure  
et vérifier si il est possible de la décoder en réception



Envoie message CAN code:

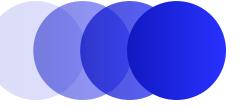
```
// Envoi des données par CAN
void ModuleMesure::envoyerDonnees() {
    if (_directionVent < 1 || _directionVent > 8) {
        Serial.println("Erreur: Données de direction invalides, envoi annulé");
        return;
    }

    Serial.print("Envoi des données par CAN avec ID 0x");
    Serial.print(_canId, HEX);
    Serial.print("... ");

    CAN.beginPacket(_canId);
    CAN.write((uint8_t)_moduleId);
    CAN.write((uint8_t)_vitesseVent);
    CAN.write((uint8_t)_directionVent);

    bool success = CAN.endPacket();

    if (success) {
        Serial.println("terminé");
        _dernierTempsEnvoi = millis();
    } else {
        Serial.println("ÉCHEC");
        delay(100);
    }
}
```



Test Unitaire CAN et vérifier le bon envoie d'un message

## Composition Trame CAN

### Envoie

--- Données du Module 1 ---  
Module ID: 1 ←  
Vitesse du vent: 9 km/h ←  
Direction du vent: 2  
Envoi des données par CAN avec ID 0x100... terminé ←

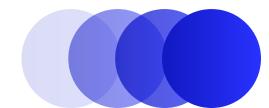
### Réception

→ -----  
→ Data Reçu:  
→ -----  
→ Numéro Module: 1  
→ Vitesse Vent: 9 km/h  
→ Direction: 2  
→ -----  
→ -----

Résultat Concluant

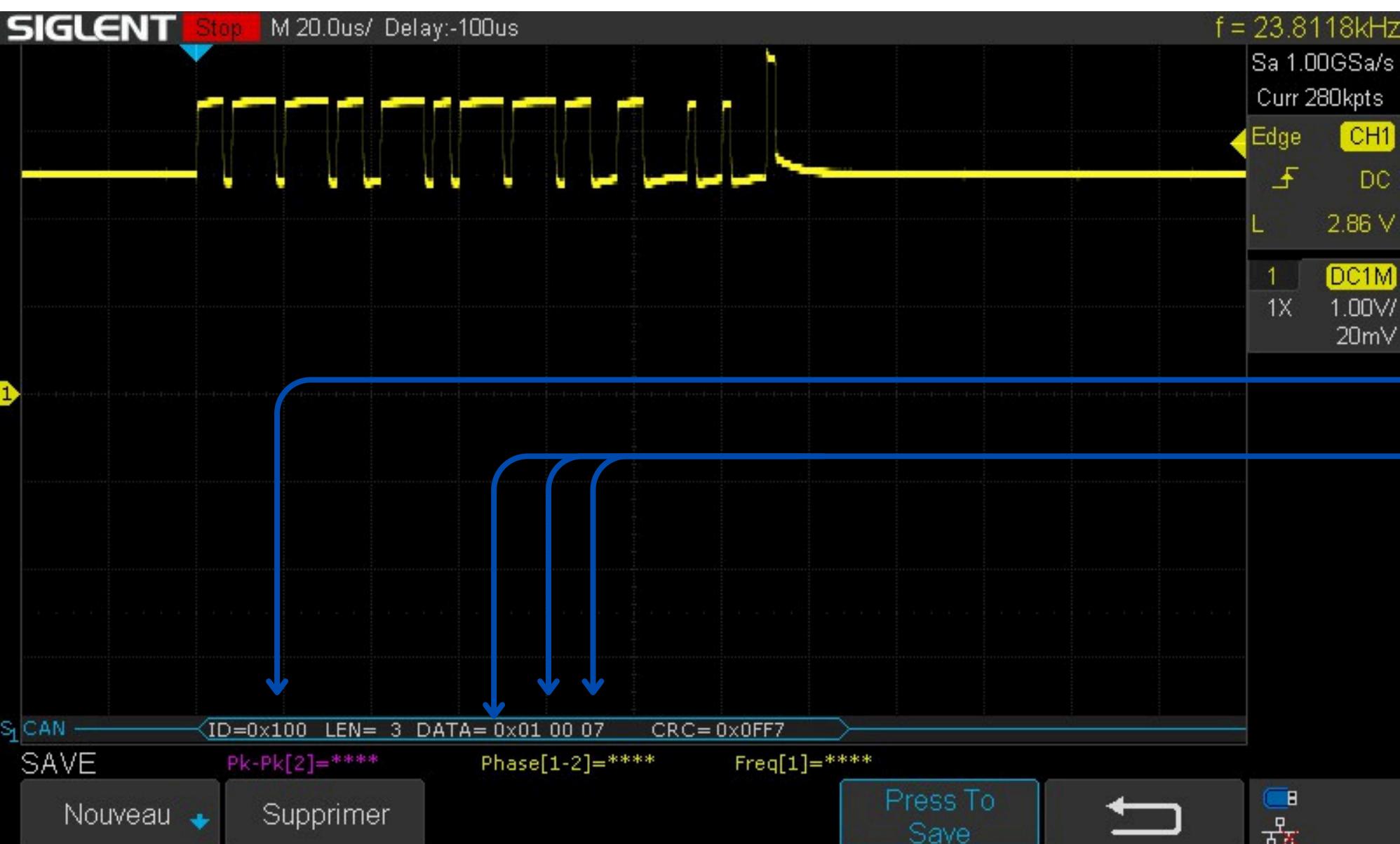
Champ	Taille (bits)
<b>Start of Frame</b>	1
<b>Arbitration Field</b>	12
<b>Control Field</b>	6
<b>Data Field</b>	0-64 (0-8 octets) Dans notre cas 3 oct
<b>CRC Field</b>	15 + 1
<b>ACK Field</b>	2
<b>End of Frame</b>	7
<b>Intermission</b>	3 (non inclus dans la trame)

## 02 Présentation de la conception



Test Unitaire CAN et vérifier le bon envoi d'un message via oscilloscope

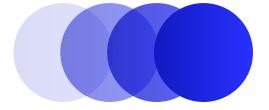
### Composition Trame CAN



Champ	Taille (bits)
<b>Start of Frame</b>	1
<b>Arbitration Field</b>	12
<b>Control Field</b>	6
<b>Data Field</b> 0-64 (0-8 octets) Dans notre cas 3 oct	0-64 (0-8 octets) Dans notre cas 3 oct
<b>CRC Field</b>	15 + 1
<b>ACK Field</b>	2
<b>End of Frame</b>	7
<b>Intermission</b>	3 (non inclus dans la trame)

05

## Conclusion personnelle sur l'état d'avancement



### Probleme 1

01

Problème Matériel ou logiciel qui bloquant le téléchargement du code sur l'esp 32 retardant les tests inréalisable

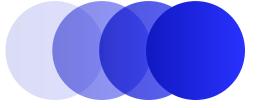
### Probleme 2

02

Problème de Bibliothèque ESP 32 bloquant la communication BUS CAN

05

## Conclusion personnelle sur l'état d'avancement



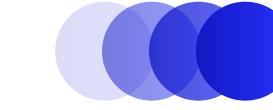
# Futures Objectifs

01

Réaliser l'application  
QT pour le technicien

02

Mettre en relation la  
partie Acquisition avec  
l'armoire centrale



**Merci pour  
votre attention**