

BTS CIEL

Épreuve E6.2 – Session 2026

Dossier de revue finale

Projet Biathlon

Lucas SPINOSI-
BAUMET (IR)
Youssef EL
ALLOUCHI (IR)
Thomas
RETAMOS (IR)
Alexandre
PARISIS (ER)
Timéo KLIPPEL
(ER)

Partie commune

Table des matières

1. Introduction et Contexte du Projet	4
2. Architecture Globale.....	4
3. Répartition des Tâches au sein de l'Équipe.....	5
Pôle Informatique et Réseaux (IR)	5
Pôle Électronique et Réseaux (ER)	5
4. Cahier des charges	6
5 - Audit de sécurité.....	6

1. Introduction et Contexte du Projet

Le projet Biathlon vise à concevoir un équipement de tir sportif électronique en réponse à la demande de Monsieur LeDuc, professeur d'EPS, afin de proposer une activité de biathlon comme épreuve au baccalauréat.

L'objectif principal est de remplacer les munitions réelles par un système laser sécurisé, tout en maintenant les sensations et les contraintes du tir de compétition. Le système a été pensé pour être simple d'utilisation, peu coûteux, proche du matériel professionnel et facilement reproductible grâce aux fichiers de fabrication.

Techniquement, le projet consiste à numériser et fiabiliser la détection de tirs laser sur des cibles physiques, tout en centralisant les données (scores, état du matériel) sur un superviseur.

2. Architecture Globale

Le système complet s'articule autour de trois éléments principaux:

- **La Carabine Laser** : L'équipement du tireur, chargé d'émettre le tir laser et de communiquer son état.



- **La Cible Électronique** : Le récepteur physique qui détecte les impacts laser et gère les affichages visuels (LEDs).



- **Le Superviseur (Logiciel/Interface) :** Le cerveau central qui arbitre la partie, gère les modes de jeu et stocke les résultats en base de données.



3. Répartition des Tâches au sein de l'Équipe

Pour mener à bien cette refonte complète (version 2026), l'équipe a été divisée en deux pôles distincts : l'Informatique et Réseaux (IR) pour la partie logicielle, et l'Électronique et Réseaux (ER) pour la conception matérielle de l'architecture physique.

Pôle Informatique et Réseaux (IR)

- **Lucas Spinosi-Baumet (Développement Logiciel Cible) :** En charge de la programmation du microcontrôleur ESP32 de la cible. Ses missions incluent le passage du code en Programmation Orientée Objet (POO), l'optimisation du protocole TCP pour réduire la latence, la gestion matérielle autonome (monitoring de batterie), et le développement de l'algorithme de réconciliation d'état avec le superviseur.
- **Thomas Retamosa (Développement Logiciel Carabine) :** Responsable du code de la carabine sur ESP32. Il gère l'émission du tir laser via le protocole infrarouge NEC, la communication TCP/IP avec le superviseur, l'acquisition de la tension de la batterie, et la gestion des différents états de l'arme (modes Entraînement et Examen).
- **Youssef El Allouchi (Superviseur Qt et Base de Données) :** Concepteur de l'application centrale développée en C++ (Qt). Il gère le serveur TCP asynchrone capable de communiquer avec plusieurs équipements simultanément, la base de données MySQL pour l'enregistrement des scores, et l'arbitrage automatique (algorithme de synchronisation) en cas de coupure réseau.

Pôle Électronique et Réseaux (ER)

- **Alexandre Parisi (Conception Matérielle Carabine) :** En charge de la nouvelle architecture électronique de la carabine. Ses tâches comprennent la miniaturisation du PCB pour l'intégrer au nouveau boîtier, la suppression des composants inutiles de l'ancienne version, l'ajout d'une LED bicolore pour le niveau de batterie, et le passage à une alimentation par batteries LiPo.
- **Timéo Klippel (Conception Matérielle Cible) :** Responsable de la refonte de la carte électronique de la cible. Il gère l'optimisation du schéma structurel, le routage du PCB sur KiCad, le multiplexage des LEDs d'affichage, ainsi que l'intégration et la

soudure d'une nouvelle antenne Wi-Fi (XIAO ESP32-S3) pour fiabiliser la portée réseau en extérieur.

4.Cahier des charges

Il pourra y avoir entre 1 et 8 (paramétrable) coureurs simultanément.

Avant l'épreuve, le professeur (arbitre) paramètre le déroulement de la course et démarre la session (chronomètre).

L'épreuve commence par une course autour du stade (pas loin de 2 tours).

Arrivé au poste de tir, le coureur se munit d'un pistolet laser.

En cas de faute de la part du coureur pendant la session (virage coupé par exemple), le juge qui le suit peut lui attribuer une pénalité de 30 secondes (valeur paramétrée par l'arbitre avant le début de la session).

Après concentration, cinq tirs (paramétrable) sont exécutés sur une cible personnelle.

Un signal lumineux permettra au coureur de savoir si la cible est atteinte.

Le système affiche les temps aux tours avant passage au stand de tir, après, avec et sans pénalité et rend disponible le résultat pour le public (Page web public).

Lorsque tous les coureurs ont terminé, l'arbitre met fin à la course, et a la possibilité de sauver les résultats sur un support externe.

Initialement, nous étions soumis à différents types de contraintes :

- Contraintes financières (budget alloué) : 500€ max
- Contraintes de développement (matériel et/ou logiciel imposé / technologies utilisées) :
- Contraintes qualité (conformité, délais, ...) : Les matériels pistolet et cible doivent ressembler à la réalité.
- Contraintes de fiabilité, cyber sécurité : Le système doit être protégé contre les attaques (WIFI), les données doivent être transportées avec sécurité et fiabilité en utilisant des protocoles chiffrés. Un pare-feu doit être paramétré pour sécuriser les communications.

5 - Audit de sécurité

Fiche d'audit de sécurité (Réf. 2026-1)

Projet	Auteur	Date
Projet Biathlon	Lucas Spinosi-Baumet Youssef El Allouchi Thomas Retamosa	30/05/26

Désignation	Risque	Solution
Accès WEB	• Intrusion • Injection de requête SQL • Lecture de trames	• Site en HTTPS • Sécuriser les formulaires • Sécuriser les droits des fichiers serveurs
SGBD	• Attaque pour trouver le mot de passe admin • Piratage des données	• Mot de passe complexe du SGBD • Chiffrement du mot de passe SGBD
WIFI	• Déchiffrer le mot de passe	• Utilisation WPA2
Carabine et cible	• Utilisation d'un autre laser pour éviter les pénalités	• Laser contenant une trame spécifique
Communication	• Lire les trames entre équipements grâce à wireshark	• Chiffrer les trames avec l'utilisation du OU exclusif

Synthèse

L'audit de sécurité du projet Biathlon met en évidence plusieurs risques sur les plans web, réseau, base de données et matériel. Des solutions adaptées ont été proposées pour chaque point sensible : sécurisation HTTPS, protection des formulaires, mots de passe robustes, et authentification des dispositifs de tir. Ces mesures permettent de renforcer la fiabilité et l'intégrité du système dans son ensemble

Lucas Spinosi-Baumet

Partie Personnelle- Lucas Spinosi-Baumet

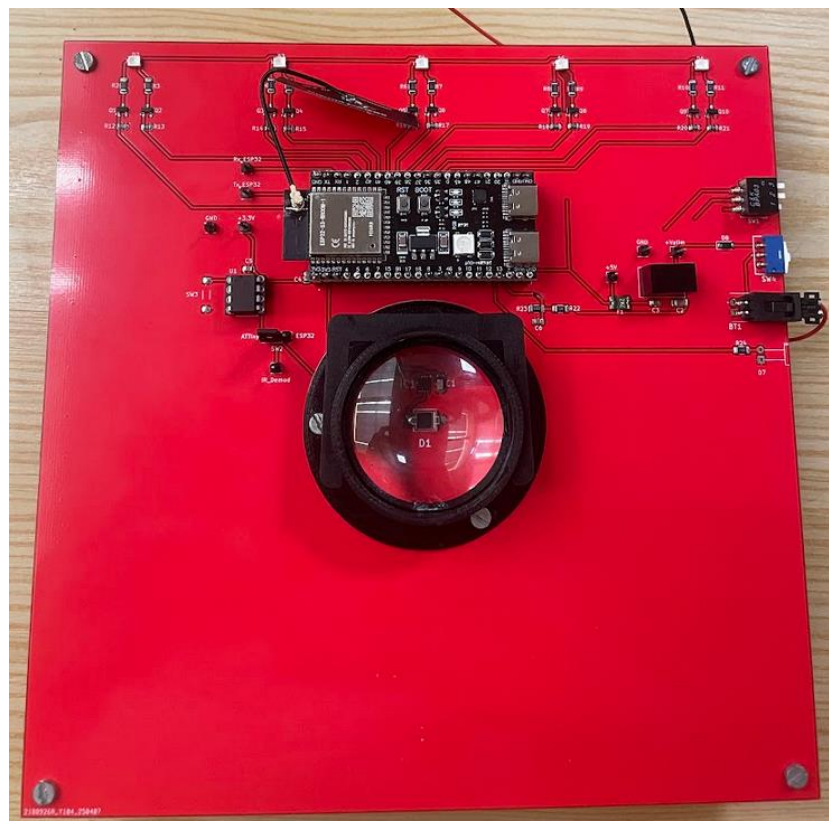


TABLE DES MATIÈRES

Lycée Alphonse Benoît	1
1. Architecture matérielle	3
2. Introduction et objectifs de la mise à jour	3
3. Optimisation Réseau et Fiabilisation des Échanges TCP	3
I. Choix du protocole : Pourquoi TCP ?	4
II. La sécurisation des échanges TCP (Chiffrement XOR)	4
III. Réduction drastique de la latence : Désactivation de l'algorithme de Nagle	4
IV. Stratégies de reconnexion non-bloquantes	4
4. Transition du code en Programmation Orientée Objets	5
5. Gestion des deux modes, cible avec 2 états différents	6
I. Mode Entraînement (Autonome) :	7
II. Mode Examen (Supervisé) :	7
6. Le Multiplexage des affichages visuels (LedsCible)	8
7. Monitoring batterie	9
8. Mise en place d'un système de synchronisation avancée	10
9. L'adressage matériel dynamique (Classe DipSwitch)	12
10. Diagrammes de Séquence de la cible et de la Synchronisation	13
11. Scénarios de Tests et Validation du Système	13
12. Protocole de communication	14
I. Trames : Cible vers Superviseur	14
II. Trames : Superviseur vers Cible	15
13. Etude physique : Analyse de l'antenne	16
14. Calendrier prévisionnel et réel	17
15. Conclusion	18
16. Annexes	20
I. Diagramme de séquence du fonctionnement global de la cible	20
II. Diagramme de séquence du fonctionnement de la synchronisation de la cible	21

1. Architecture matérielle

Le cœur de chaque cible physique est architecturé autour d'un microcontrôleur ESP32-S3



2. Introduction et objectifs de la mise à jour

Le projet Biathlon vise à numériser et fiabiliser la détection de tirs laser sur des cibles physiques, tout en centralisant les scores sur un superviseur logiciel (Qt). Lors des phases de test précédentes, plusieurs vulnérabilités ont été identifiées, notamment liées à la stabilité du réseau Wi-Fi, à l'organisation du code, des conflits dans le code, et à la gestion matérielle de l'énergie. Les trois axes d'amélioration majeurs sont :

- La modularité par l'utilisation de la Programmation Orientée Objet (POO).
- La résilience réseau grâce à un algorithme de "Réconciliation d'État" entre la cible, la carabine et le superviseur.
- La gestion matérielle autonome (monitoring de batterie et fonctionnement hors-ligne).

3. Optimisation Réseau et Fiabilisation des Échanges TCP

La communication entre les cibles, les carabines et le superviseur constitue le cœur du projet Biathlon. Un retard de transmission ou une perte de paquet peut fausser un résultat lors de l'évaluation finale. Cette section détaille les optimisations protocolaires mises en place pour garantir une réactivité maximale.

I. Choix du protocole : Pourquoi TCP ?

Le choix s'est porté sur le protocole TCP (Transmission Control Protocol) plutôt que l'UDP. Bien que l'UDP soit plus rapide, il ne garantit pas la livraison des paquets. Dans notre contexte, chaque trame « H » (impact réussi) doit être obligatoirement envoyée. Le protocole TCP assure que chaque message est acquitté, et gère automatiquement les réémissions en cas de collision sur le réseau Wi-Fi.

II. La sécurisation des échanges TCP (Chiffrement XOR)

Afin d'éviter qu'un utilisateur malveillant connecté au réseau puisse intercepter ou falsifier les scores (trames en clair), j'ai implémenté un chiffrement symétrique par algorithme XOR. Les données sont chiffrées à l'aide d'une clé secrète partagée (« TIR »). La méthode `chiffrerXOR` convertit chaque caractère de la trame, applique le masque XOR, puis transforme le résultat en valeurs hexadécimales avant l'envoi via TCP. À la réception, la méthode `dechiffrerXOR` fait l'opération inverse. Ainsi, même si le trafic Wi-Fi est écouté, les trames comme « H1 » ou « TS3:101 » sont totalement masquées.

III. Réduction drastique de la latence : Désactivation de l'algorithme de Nagle

Par défaut, TCP tente d'optimiser la bande passante en regroupant les petits messages avant de les envoyer (Algorithme de Nagle). Pour le projet biathlon, ce comportement n'est pas adapté car il introduit un délai entre le tir et l'affichage sur le superviseur.

Optimisation « NoDelay » : Dans la classe `Reseau`, j'ai forcé la désactivation de ce regroupement lors de la connexion : `client.setNoDelay(true);`.

Cette ligne est fondamentale : elle force l'envoi immédiat de chaque trame dès que la fonction `envoyer()` est appelée. Cela garantit que l'impact laser détecté sur la cible est transmis au superviseur en quelques millisecondes, permettant une mise à jour de l'IHM quasiment instantanée pour les spectateurs et les arbitres.

IV. Stratégies de reconnexion non-bloquantes

Un des défis majeurs rencontrés était le blocage du programme lors d'une perte de signal. Si la cible attend indéfiniment de retrouver le serveur, elle ne peut plus détecter les tirs laser.

Distinction entre `connecter()` et `reconnecter()` : J'ai implémenté deux logiques distinctes dans `Reseau.cpp` :

- Au démarrage (`connecter`) : Le système est patient et effectue jusqu'à 20 tentatives (soit environ 10 secondes d'attente) pour s'assurer que l'infrastructure est prête.
- En cours de match (`reconnecter`) : La priorité est de ne pas geler la cible. Le nombre de tentatives est réduit à 2 seulement.

```

bool Reseau::reconnecter() {
    Serial.println("\n--- TENTATIVE CONNEXION WIFI ---");
    if (WiFi.status() != WL_CONNECTED) {
        WiFi.mode(WIFI_STA);
        WiFi.disconnect();
        delay(100);
        WiFi.begin(ssid, password);
    }
    int decomptereco = 0;
    while (WiFi.status() != WL_CONNECTED && decomptereco < 2) {
        delay(500); Serial.print("."); decomptereco++;
    }
}

```

Dans le `loop()` principal de la cible, cette fonction n'est appelée qu'une fois par seconde grâce à un timer (`millis() - timerReconnexion >= 1000`). Cela permet à la cible de rester réactive au laser même pendant qu'elle cherche à retrouver son serveur.

4. Transition du code en Programmation Orientée Objets

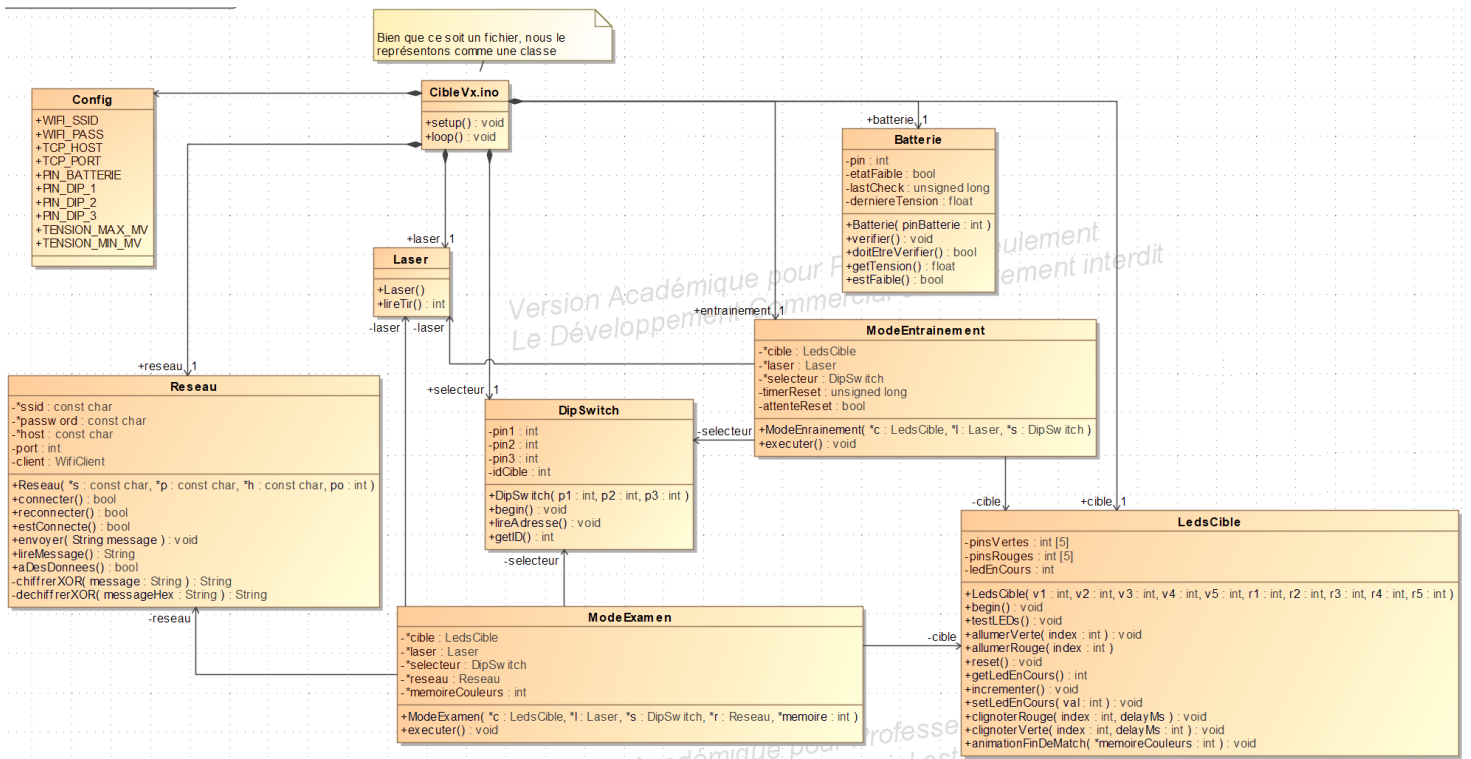
L'allègement du code principal (`CibleVx.ino`) a été poussé à son maximum lors de la dernière version logicielle. Les variables (mots de passe WiFi, adresses IP, seuils de batterie, numéros de broches) ont été externalisées dans un fichier `Config.h`. De plus, la logique d'état a été totalement isolée dans deux classes distinctes : `ModeEntrainement` et `ModeExamen`. Le programme principal ne fait plus que lire le réseau et instancier ces objets, le rendant extrêmement lisible.

Les classes développées sont les suivantes :

- `LedsCible` : Gère le multiplexage et les animations (Vert/Rouge) du plateau.
- `Laser` : Isole la logique de lecture des données du capteur série.
- `Reseau` : Encapsule la complexité du client WiFi et de l'envoi/réception TCP. Elle intègre une méthode `reconnecter()` silencieuse pour ne pas bloquer le programme.
- `DipSwitch` : Gère la configuration matérielle de l'ID de l'équipement.
- `Batterie` : Protège la durée de vie du composant matériel.
- `ModeExamen` : Gère le mode Examen

- ```
#include <Arduino.h>
#include "Config.h"
#include "DipSwitch.h"
#include "LedsCible.h"
#include "Batterie.h"
#include "Reseau.h"
#include "Laser.h"
#include "ModeEntrainement.h"
#include "ModeExamen.h"
```

### Diagramme des classes du code de la cible



## 5. Gestion des deux modes, cible avec 2 états différents

Pour qu'une cible soit utilisable même en l'absence du serveur central, le système intègre désormais une machine à états à deux modes, basculable dynamiquement.

### I. Mode Entraînement (Autonome) :

Ce mode s'active si la cible échoue à se connecter au Wi-Fi au démarrage, ou sur ordre du superviseur. La cible détecte les tirs, allume ses LEDs vertes, et se réinitialise automatiquement au bout de 5 tirs grâce à la classe

`ModeEntraînement`. Le superviseur n'intervient pas.

### II. Mode Examen (Supervisé) :

Mode par défaut lors de la connexion. La cible devient "esclave". Elle transmet ses détections laser (Trame `H`) et affiche les mauvais tirs que si le serveur lui ordonne.

**Sécurité et Auto-Reset en fin de match :** Lors de nos tests de robustesse, un nouveau scénario de blocage a été identifié : si une cible termine sa série (5 LEDs allumées) mais que le superviseur Qt plante ou subit une micro-coupure réseau avant d'envoyer la commande "RESET", la cible restait bloquée indéfiniment. Pour corriger cela, j'ai introduit un "Timer" d'attente passif. Dès que les 5 LEDs sont allumées, un chronomètre (`timerAttenteReset`) se lance. Si le superviseur n'a pas acquitté la fin de série au bout de 60 secondes (ou si la connexion est perdue), la cible déclenche son animation de fin de match et force un redémarrage d'elle-même (Auto-Reset).

Voici comment est effectué le changement d'état quand la cible est connectée au superviseur. C'est le superviseur qui envoie l'ordre à la cible pour changer d'état.

```

if (message.indexOf("MT") >= 0) {
 if (modeEntrainement == true) {
 Serial.println("La cible est déjà en mode ENTRAINEMENT");
 }
 else {
 Serial.println(">>> PASSAGE EN MODE : ENTRAINEMENT (Via TCP)");
 modeEntrainement = true;
 cible.reset();
 for(int i=0; i<5; i++) memoireCouleurs[i] = 0;
 }
}

else if (message.indexOf("ME") >= 0) {
 if (modeEntrainement == false) {
 Serial.println("La cible est déjà en mode EXAMEN");
 }
 else {
 Serial.println(">>> PASSAGE EN MODE : EXAMEN (Via TCP)");
 modeEntrainement = false;
 cible.reset();
 for(int i=0; i<5; i++) memoireCouleurs[i] = 0;
 }
}
}

```

## 6. Le Multiplexage des affichages visuels (LedsCible)

La cible contient 10 LEDs (5 vertes et 5 rouges). Les piloter une par une de manière séquentielle produirait un code interminable (une suite de 10 `digitalWrite` pour chaque animation).

Pour rendre le code évolutif, j'ai utilisé des tableaux en C++. Les broches sont stockées dans deux tableaux dès l'initialisation de l'objet :

- `const int pinsV[5] = {2, 41, 39, 37, 35};`
- `const int pinsR[5] = {1, 42, 40, 38, 36};`

Cela permet de créer des boucles `for` très courtes pour gérer l'affichage. Par exemple, la fonction d'animation de fin de match (lorsque la cible a validé ses 5 tirs) illustre parfaitement ce concept :

Dans cette fonction, on observe que le balayage visuel se fait sans effort : on éteint tout, puis on allume séquentiellement soit la LED verte, soit la LED rouge en fonction de ce qui a été mémorisé dans le tableau `memoireCouleurs[i]`. L'utilisation de boucles imbriquées (`for (int cycle = 0; cycle < 5; cycle++)`) contenant les vérifications des 5 LEDs) permet de créer des clignotements complexes en seulement une dizaine de lignes de code.

```

void animationFinDeMatch() {
 //Serial.println(">> 5 LEDS ALLUMÉES : FIN DE SÉRIE !");
 delay(5000);

 for (int cycle = 0; cycle < 5; cycle++) {
 for (int i=0; i<5; i++) {
 digitalWrite(pinsV[i], LOW);
 digitalWrite(pinsR[i], LOW);
 }
 delay(500);
 for (int i=0; i<5; i++) {
 if (memoireCouleurs[i] == 1) digitalWrite(pinsV[i], HIGH);
 if (memoireCouleurs[i] == 2) digitalWrite(pinsR[i], HIGH);
 }
 delay(500);
 }

 cible.reset();
 for(int i=0; i<5; i++) memoireCouleurs[i] = 0;
 //Serial.println(">> PLATEAU RÉINITIALISÉ");
}

```

## 7. Monitoring batterie

Dans un contexte sportif, une cible qui s'éteint sans prévenir peut annuler une compétition. La classe **Batterie** a été créée pour surveiller la tension d'alimentation quand la cible est allumée.

- Contrôle au Boot : Au démarrage, la tension est lue. Le plateau affiche le statut avec un code couleur : 5 LEDs Vertes (>12.1V), 5 Oranges (>11.2V), ou 5 Rouges (Faible). Le statut est remonté au serveur par les trames **TH**, **TM** ou **TL**.
- Contrôle cyclique : Toutes les 30 secondes, une lecture ADC est effectuée sur 10 échantillons pour lisser les parasites de mesure. Si la tension devient trop basse en pleine partie, une alerte est envoyée au superviseur.

Voici comment se passe l'acquisition de la tension de la batterie :

```

void Batterie::verifier() {
 uint32_t Valeur_ADC = 0;
 for (int i = 0; i < 10; i++) {
 Valeur_ADC += analogReadMilliVolts(pin);
 delay(2);
 }
 Valeur_ADC = Valeur_ADC / 10;

 derniereTension = Valeur_ADC * 4.636;
 etatFaible = (derniereTension < 11200);

 lastCheck = millis();
}

```

La lecture d'une tension de batterie (12V) par un microcontrôleur fonctionnant en 3.3V impose l'utilisation d'un Convertisseur Analogique-Numérique (ADC). Cependant, les ADC des ESP32 sont connus pour être particulièrement sensibles au bruit électronique, ce qui peut causer des fluctuations de lecture et déclencher de fausses alertes de batterie faible.

Pour résoudre ce problème, j'ai mis en place deux traitements mathématiques dans mon code :

Le lissage par moyenne glissante (Filtre logiciel) : Plutôt que de faire une seule lecture `analogReadMilliVolts()`, l'algorithme réalise une boucle de 10 itérations espacées de 2 millisecondes. Toutes les valeurs sont additionnées puis divisées par 10. Ce filtre passe-bas logiciel permet de gommer les pics parasites et d'obtenir une tension de batterie parfaitement stable.

Le coefficient de conversion (4.636) : La tension renvoyée par le code n'est pas la tension directe, mais une valeur passée au travers d'un pont diviseur de tension (pour abaisser les 12V de la batterie à une plage acceptable pour l'ESP32). Le coefficient 4.636 utilisé dans le code (`derniereTension = Valeur_ADC * 4.636;`) a été calculé théoriquement pour inverser l'effet du pont diviseur et retrouver la tension réelle de la batterie au millivolt près.

**Voici comment se passe l'envoi de l'état de la batterie :**

```

if (tension >= 12100) {
 // Niveau 3 : PLEINE -> 5 Vertes
 for(int i=0; i<5; i++) cible.allumerVerte(i);
 msgBatterie = "TH" + String(selecteur.getID()) + "\n";
}
else if (tension >= 11200) {
 // Niveau 2 : MOYENNE -> 5 Oranges (Rouge + Vert)
 for(int i=0; i<5; i++) {
 cible.allumerVerte(i);
 cible.allumerRouge(i);
 }
 msgBatterie = "TM" + String(selecteur.getID()) + "\n";
}
else {
 // Niveau 1 : FAIBLE -> 5 Rouges
 for(int i=0; i<5; i++) cible.allumerRouge(i);
 msgBatterie = "TL" + String(selecteur.getID()) + "\n";
}

reseau.envoyer(msgBatterie + String(selecteur.getID()) + "\n");

```

## 8. Mise en place d'un système de synchronisation avancée

Le défi principal dans ce système (Cible + Carabine + Serveur) est la perte de paquets en cas de déconnexion . Par exemple, si la carabine perd le réseau et que le joueur rate la cible, le serveur n'est jamais mis au courant. Pour pallier cela, j'ai pensé un algorithme de Réconciliation d'État côté superviseur (Qt) que mon collègue Youssef à développé.

Le système des vérités sont les suivantes :

- La Carabine est la source de vérité pour le *nombre total de tirs*.
- La Cible est la source de vérité pour les *touches réussies*.

**Algorithme de traitement :**

- 1) Lors d'une reconnexion matérielle, le serveur Qt (Superviseur) émet une requête de synchronisation globale : la trame **S**.
- 2) La cible répond avec la séquence de ses LEDs actuelles (ex pour la cible 3: **TS3:101** pour Vert-Rouge-Vert).
- 3) La carabine répond avec son nombre de tirs totaux (ex pour la carabine 3:

TC3:4).

- 4) Le superviseur reçoit d'abord la vérité des bons tirs de la cible et stocke les succès (les "1"). Ensuite, il compare la taille de la séquence avec le nombre total de tirs de la carabine. S'il y a un décalage (ex: carabine = 4 tirs, séquence cible = 3 tirs), le serveur déduit immédiatement que les tirs fantômes survenus hors-ligne étaient des ratés, et complète la séquence avec des 0. La Base de données est alors corrigée et la vérité absolue est renvoyée aux composants physiques.

```
// -----> AJOUT DE LA DEMANDE DE SYNCHRONISATION <-----
else if (message.indexOf("S") >= 0) {
 Serial.println(">>> DEMANDE DE SYNCHRO RECUE, ENVOI DE L'ETAT LOCAL...");

 String seqLocal = "";
 // On lit ce qu'on a actuellement en mémoire (1=Vert, 2=Rouge)
 for(int i = 0; i < cible.getLedEnCours(); i++) {
 if (memoireCouleurs[i] == 1) seqLocal += "1";
 else if (memoireCouleurs[i] == 2) seqLocal += "0";
 }

 // On envoie la trame TSYNC au superviseur
 String msgSync = "TS" + String(selecteur.getID()) + ":" + seqLocal + "\n";
 reseau.envoyer(msgSync);
}
```

```
// Si c'est bien une trame de synchronisation
if (isSync && message.length() > 0 && message.length() <= 5) {
 //Serial.println(">>> SYNCHRONISATION DU PLATEAU : " + message);

 cible.reset();
 for(int i=0; i<5; i++) memoireCouleurs[i] = 0;

 for (int i = 0; i < message.length(); i++) {
 if (message[i] == '1') {
 memoireCouleurs[i] = 1;
 cible.allumerVerte(i);
 } else if (message[i] == '0') {
 memoireCouleurs[i] = 2;
 cible.allumerRouge(i);
 }
 }

 cible.setLedEnCours(message.length());

 if (message.length() == 5) {
 animationFinDeMatch();
 }
}
```

## 9. L'adressage matériel dynamique (Classe DipSwitch)

Dans un système comportant plusieurs cibles, le superviseur doit pouvoir les différencier. Plutôt que de "coder en dur" un numéro dans chaque ESP32 (ce qui obligerait à téléverser un code différent pour chaque cible), j'ai opté pour un adressage matériel via un DipSwitch à

3 interrupteurs.

Configuration des broches : Dans la méthode `begin()`, les broches sont initialisées en mode `INPUT_PULLUP`. Ce choix est crucial : il active les résistances de tirage internes du microcontrôleur ESP32. Cela permet d'éviter les "états flottants" (bruit électrique) lorsque l'interrupteur est ouvert, et d'économiser l'ajout de 3 résistances externes sur notre circuit imprimé (PCB).

Algorithme de conversion Binaire vers Décimal : L'état des 3 interrupteurs forme un mot binaire de 3 bits, permettant d'adresser jusqu'à 8 cibles différentes (de 0 à 7). J'ai écrit un algorithme simple de conversion dans la méthode `lireAdresse()` :

```
void DipSwitch::lireAdresse() {
 idCible = 0;
 if (!digitalRead(pin1)) idCible += 1;
 if (!digitalRead(pin2)) idCible += 2;
 if (!digitalRead(pin3)) idCible += 4;

 Serial.println("-----");
 Serial.printf(" CIBLE CONFIGURÉE ID : %d \n", idCible);
 Serial.println("-----");
}
```

Le signal étant lu en `PULLUP`, un interrupteur fermé renvoie `LOW` (0). L'algorithme vérifie chaque bit :

- Le Pin 1 représente le bit de poids faible ( $2^0 = 1$ ).
- Le Pin 2 représente ( $2^1 = 2$ ).
- Le Pin 3 représente le bit de poids fort ( $2^2 = 4$ ).

Par de simples additions, l'ID de la cible est calculé dynamiquement au démarrage, rendant le code 100% générique pour toutes les cibles de la compétition.

## **10. Diagrammes de Séquence de la cible et de la Synchronisation**

Fonctionnement Global de la cible (*voir annexe I*)

Reconnexion et Réconciliation d'état (*voir annexe II*)

# 11. Scénarios de Tests et Validation du Système

Pour garantir la fiabilité du système lors d'une compétition de biathlon, une série de tests a été menée. L'objectif était de soumettre l'architecture matérielle et logicielle à des cas d'usage extrêmes (pertes d'alimentation, coupures réseau, désynchronisation) afin de valider la robustesse des algorithmes développés.

Les tests ont été structurés autour des fonctionnalités critiques. Le tableau ci-dessous résume les scénarios de validation majeurs :

## Plan de Tests d'Intégration Globale

| ID Test | Fonctionnalité Testée                     | Conditions Initiales                               | Action / Événement                                             | Résultat Attendu                                                                                                                            | Statut |
|---------|-------------------------------------------|----------------------------------------------------|----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|--------|
| TEST-01 | Autonomie                                 | Cible allumée, connectée au serveur (Mode Examen). | Coupure volontaire du routeur Wi-Fi.                           | Détection de la perte. Maintien du Mode Examen. Maintien de la détection laser.                                                             | Validé |
| TEST-02 | Réconciliation d'état (Raté hors-ligne)   | Cible et Carabine déconnectées du serveur.         | Le tireur tire à côté de la cible (Raté).                      | À la reconnexion, le serveur constate la différence entre le compteur Carabine et la séquence Cible. Il insère un 0 et met à jour la cible. | Validé |
| TEST-03 | Réconciliation d'état (Touché hors-ligne) | Cible déconnectée du serveur.                      | Le tireur touche la cible (Impact laser détecté).              | La cible allume la LED verte en local. À la reconnexion, le serveur récupère les '1' et met à jour sa Base de Données.                      | Validé |
| TEST-04 | Alerte d'énergie (Monitoring)             | Cible en pleine partie (Mode Examen).              | Baisse de la tension de l'alimentation sous le seuil de 11.2V. | Envoi transparent de la trame TL<ID> au serveur sans interrompre la partie en cours.                                                        | Validé |
| TEST-05 | Forçage de mode distant                   | Cible connectée, en attente de joueurs.            | Le serveur Qt envoie la trame MT (Mode Training).              | La cible réinitialise son plateau et bascule en mode autonome ( <code>modeEntrainement = true</code> ).                                     | Validé |

L'intégralité des scénarios de tests a été passée avec succès. L'utilisation massive de logs de débogage (`Serial.println` sur l'ESP32 et `QDebug()` sur Qt) a permis de tracer

chaque événement. Le système est capable de subir des pertes d'alimentation partielles, des brouillages Wi-Fi, ou des déconnexions intempestives des joueurs, sans jamais corrompre l'intégrité de la base de données des scores de la compétition.

## 12. Protocole de communication

### I. Trames : Cible vers Superviseur

Ces messages sont générés de manière autonome par la cible et envoyés au PC. (*Note : x représente l'ID physique de la cible configuré via le DipSwitch, par exemple 1, 2, 3...*)

t = nombre de 0 ou 1

| Trame  | Événement déclencheur             | Description et Action côté Superviseur                                                                                                                           |
|--------|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TIDx\n | Connexion initiale ou Reconnexion | Identification : La cible vient de se connecter au réseau TCP. Le superviseur sait maintenant que la cible x est en ligne et prête.                              |
| Hx\n   | Tir valide détecté                | Hit (Tir réussi) : Le capteur infrarouge a reçu un code correspondant exactement à son ID. Le superviseur doit comptabiliser le point.                           |
| TLx\n  | Tension batterie < 11.2V          | Battery Low (Batterie faible) : Envoyé automatiquement toutes les 30 secondes si la cible est sur le point de s'éteindre. Le superviseur doit alerter l'arbitre. |
| TMx\n  | 11.2V <= Tension batterie < 12.1V | Battery Mid (Batterie moyenne) : Envoyé automatiquement au démarrage de la cible si la batterie est à la moitié de sa capacité.                                  |
| THx\n  | Tension batterie >= 12.1V         | Battery High (Batterie Pleine) : Envoyé automatiquement au démarrage de la cible si la batterie est pleine.                                                      |

|         |                 |                                                                                       |
|---------|-----------------|---------------------------------------------------------------------------------------|
| TSx:t\n | Synchronisation | Synchronisation de l'état des tirs : Envoi de la trame de synchronisation au serveur. |
|---------|-----------------|---------------------------------------------------------------------------------------|

## II. Trames : Superviseur vers Cible

Ces commandes sont envoyées par le PC pour piloter les cibles à distance. La cible lit les trames et cherche si le mot-clé est contenu dedans.

t = nombre de 0 ou 1

| Trame   | Nom de la commande           | Action exécutée par la Cible                                                                                                                              |
|---------|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| ME\n    | Mode Examen                  | Passage en Examen : La cible désactive son mode entraînement, réinitialise son plateau de LEDs à zéro et attend les tirs pour informer le superviseur.    |
| MT\n    | Mode Training (Entraînement) | Passage en Entraînement : réinitialise son plateau et gère ses LEDs toute seule (auto-reset après 5 tirs).                                                |
| t\n     | Synchronisation              | Synchronisation de l'état des tirs : en cas de redémarrage de la cible, le superviseur envoie une trame de synchronisation vers la cible,                 |
| R\n     | Allumer Rouge                | Tir de pénalité / Raté : Allume la LED en cours en Rouge et passe à la position suivante. Utilisé si le superviseur veut forcer l'affichage d'une erreur. |
| RESET\n | Réinitialisation             | Remise à zéro : Éteint toutes les LEDs (vertes et rouges), vide la mémoire des couleurs et replace le curseur sur la première LED.                        |

|     |                            |                                                                                                                     |
|-----|----------------------------|---------------------------------------------------------------------------------------------------------------------|
| S\n | Demande de synchronisation | Synchronisation de l'état des tirs : Envoie de la trame pour demander l'état actuel des cibles/carabines concernées |
|-----|----------------------------|---------------------------------------------------------------------------------------------------------------------|

## 13. Etude physique : Analyse de l'antenne

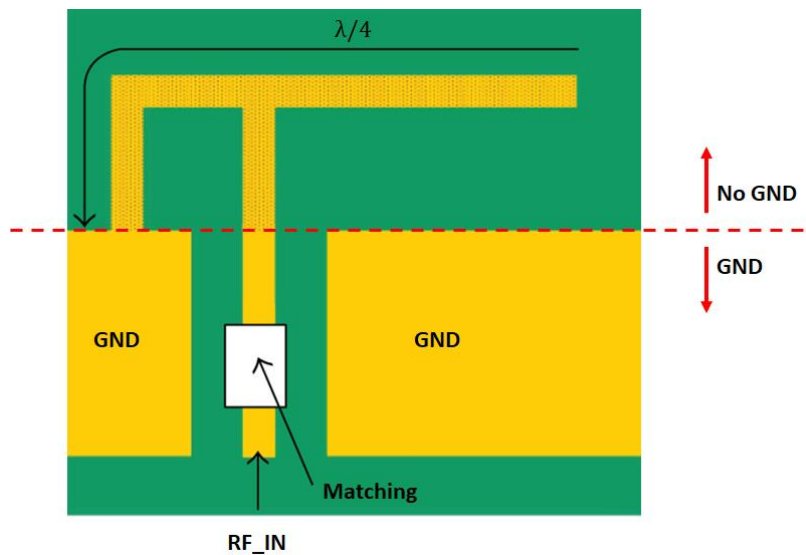
Au départ, l'ESP32S3 ne disposait pas d'une antenne suffisamment performante pour ne serait-ce garder la connexion wifi. Alors nous avons accroché sur l'antenne de base une petite pince double, ce qui a permis de pallier ce problème. Lors de nos tests en conditions réelles, nous avons mesuré la distance maximale où la cible restait connecté au superviseur (raspberry pi), on avait une distance maximale d'environ 14 mètres. Or, quand nous nous sommes entretenus avec le professeur de sport, il nous a expliqué où seraient placés les cibles, les carabines ainsi que le superviseur. Dans cette disposition là, les 14 mètres n'étaient pas du tout suffisants pour garantir un bon fonctionnement du système.

Nous avons donc ajouté une antenne, comme l'esp32s3 que j'avais ne pouvait pas nativement avoir une antenne, Mr Hortolland a effectué une manipulation sur la carte (côté antenne), en soudant un connecteur UFL CMS, pour pouvoir ensuite brancher l'antenne dessus.

Pour assurer une communication fiable, l'antenne FPC du ESP32-S3 intégrée à la carte était de base suffisante car elle répond parfaitement aux lois physiques de la transmission radio et aux contraintes matérielles du biathlon. Notre réseau Wi-Fi opère sur une fréquence de 2,4 GHz. En utilisant la vitesse d'une OEM  $c = 3,00 \cdot 10^8 \text{ m.s}^{-1}$ , la longueur d'onde  $\lambda$  se calcule ainsi :

$$\lambda = \frac{c}{f} = \frac{3 \cdot 10^8}{2.4 \cdot 10^9} = 0,125\text{m} = 12,5\text{cm}$$

Une longueur d'onde mesure donc **12,5 cm**. Afin d'optimiser la réception tout en garantissant un encombrement minimal, cette antenne utilise le principe du quart d'onde. Sa piste conductrice mesure environ un quart de la longueur d'onde complète, soit environ **3,1 cm**. Elle est donc mathématiquement et physiquement accordée à notre signal.



## 14. Calendrier prévisionnel et réel

Calendrier prévisionnel :

|  | Activité IR 2                                     | 160 hr | Lun 05/01/26 | Mer 06/05/26 |    |
|--|---------------------------------------------------|--------|--------------|--------------|----|
|  | Lecture du dossier Biathlon                       | 8 hr   | Lun 05/01/26 | Mer 07/01/26 |    |
|  | Microsoft Project                                 | 8 hr   | Mer 07/01/26 | Mar 13/01/26 | 20 |
|  | Contexte et contraintes                           | 8 hr   | Mar 13/01/26 | Lun 19/01/26 | 21 |
|  | Prise en main Logiciels + ESP32S3                 | 14 hr  | Lun 19/01/26 | Lun 26/01/26 | 22 |
|  | 1ère version simpliste (wifi + connexion en tcp)  | 20 hr  | Lun 26/01/26 | Mer 04/02/26 | 23 |
|  | Finition du programme cible                       | 32 hr  | Mer 04/02/26 | Mar 10/03/26 | 24 |
|  | Amélioration du code                              | 20 hr  | Mar 10/03/26 | Lun 23/03/26 | 25 |
|  | Recherche de failles de sécurité                  | 20 hr  | Lun 23/03/26 | Mer 01/04/26 | 26 |
|  | Déploiement du code sur les cibles + installation | 8 hr   | Mer 01/04/26 | Mar 07/04/26 | 27 |
|  | Test en conditions réelles et amélioration        | 14 hr  | Mar 07/04/26 | Lun 04/05/26 | 28 |
|  | Sujet de physique                                 | 8 hr   | Lun 04/05/26 | Mer 06/05/26 | 29 |

Calendrier réel :

|                                                                                     |                                                   |       |              |              |    |
|-------------------------------------------------------------------------------------|---------------------------------------------------|-------|--------------|--------------|----|
|    | ▲ Activité IR 2                                   | 194 h | Lun 05/01/26 | Mer 27/05/26 |    |
|    | Lecture du dossier Biathlon                       | 8 h   | Lun 05/01/26 | Mer 07/01/26 |    |
|    | Microsoft Project                                 | 8 h   | Mer 07/01/26 | Mar 13/01/26 | 19 |
|    | Contexte et contraintes                           | 8 h   | Mar 13/01/26 | Lun 19/01/26 | 20 |
|    | Prise en main Logiciels + ESP32S3                 | 14 h  | Lun 19/01/26 | Lun 26/01/26 | 21 |
|    | 1ère version simpliste (wifi + connexion en tcp)  | 12 h  | Lun 26/01/26 | Lun 02/02/26 | 22 |
|    | Finition du programme cible                       | 40 h  | Lun 02/02/26 | Mar 10/03/26 | 23 |
|    | Amélioration du code                              | 38 h  | Mar 10/03/26 | Mer 01/04/26 | 24 |
|    | Recherche de failles de sécurité                  | 20 h  | Mer 01/04/26 | Mer 15/04/26 | 25 |
|    | Déploiement du code sur les cibles + installation | 4 h   | Mer 15/04/26 | Lun 04/05/26 | 26 |
|  | Test en conditions réelles et amélioration        | 38 h  | Lun 04/05/26 | Mar 26/05/26 | 27 |
|  | Sujet de physique                                 | 4 h   | Mar 26/05/26 | Mer 27/05/26 | 28 |

## 15. Conclusion

Ce projet de mise à jour du système Biathlon 2026 avait pour objectif principal de numériser et de fiabiliser la détection des tirs laser sur des cibles physiques, tout en centralisant les scores sur un superviseur logiciel. Face aux vulnérabilités identifiées lors des phases de test précédentes, les solutions techniques apportées ont permis d'aboutir à un système robuste, modulaire et autonome.

Sur le plan logiciel, la transition vers la Programmation Orientée Objet (POO) a grandement facilité la maintenance et l'évolutivité du code de l'ESP32. La résilience du réseau a été significativement améliorée grâce au choix du protocole TCP et à la désactivation de l'algorithme de Nagle, garantissant la transmission des impacts en quelques millisecondes pour une mise à jour quasi instantanée. De plus, le développement d'un algorithme de "Réconciliation d'État" assure désormais la sauvegarde et la synchronisation des scores même en cas de déconnexion temporaire entre la cible, la carabine et le superviseur.

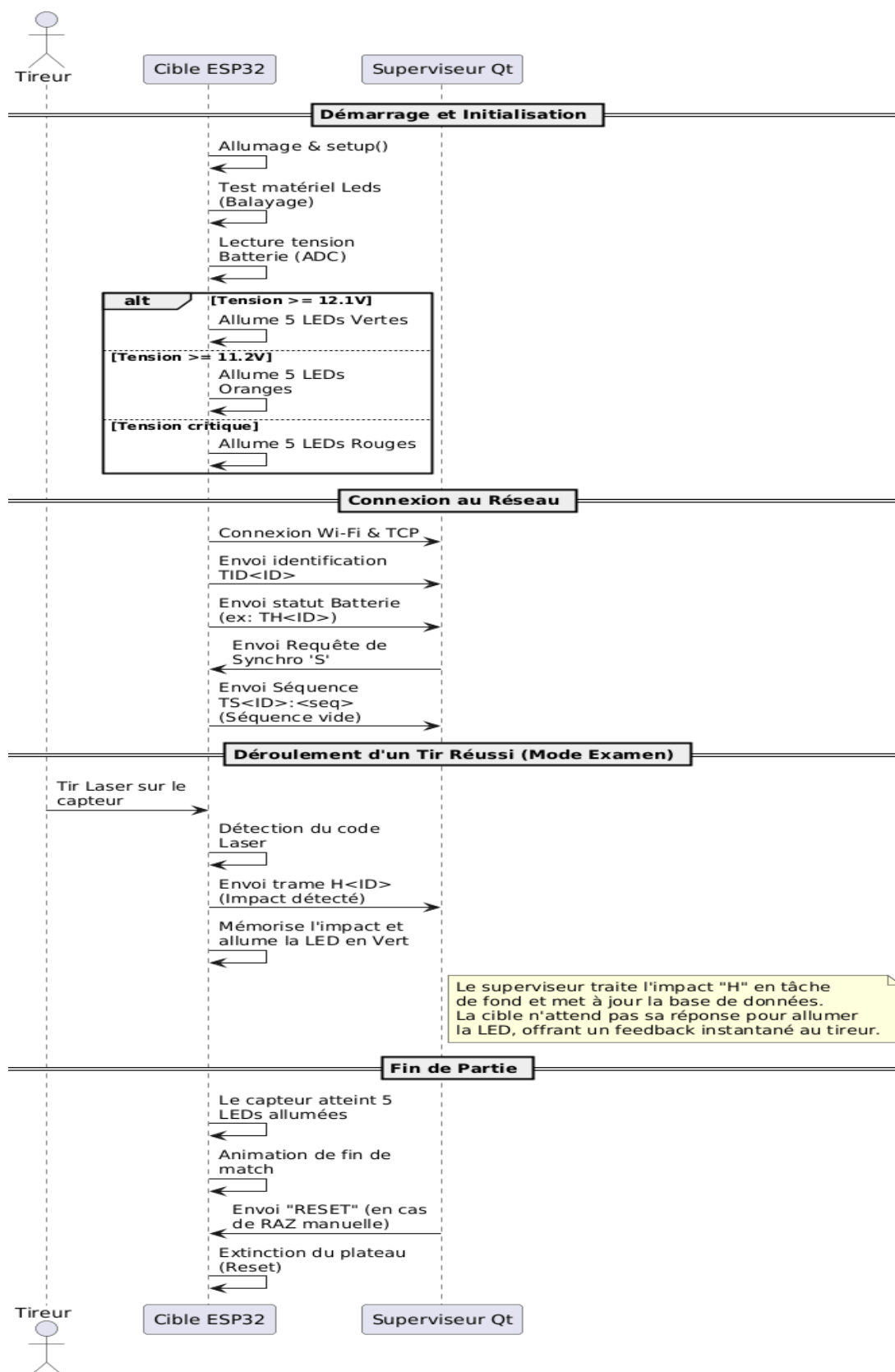
Sur le plan matériel, le DipSwitch permet un adressage dynamique simplifiant

l'intégration de plusieurs cibles pour la compétition, sans avoir à téléverser un code différent pour chacune. Le système intègre également un monitoring de batterie efficace pour prévenir les arrêts qui pourraient annuler une partie. Par ailleurs, l'étude physique a démontré la nécessité de l'intégration d'une antenne FPC exploitant le principe du quart d'onde : celle-ci a permis de pallier la limite de portée initiale de 14 mètres. Désormais, la portée maximale est d'environ 25 mètres, soit un gain de plus de 10 mètres.

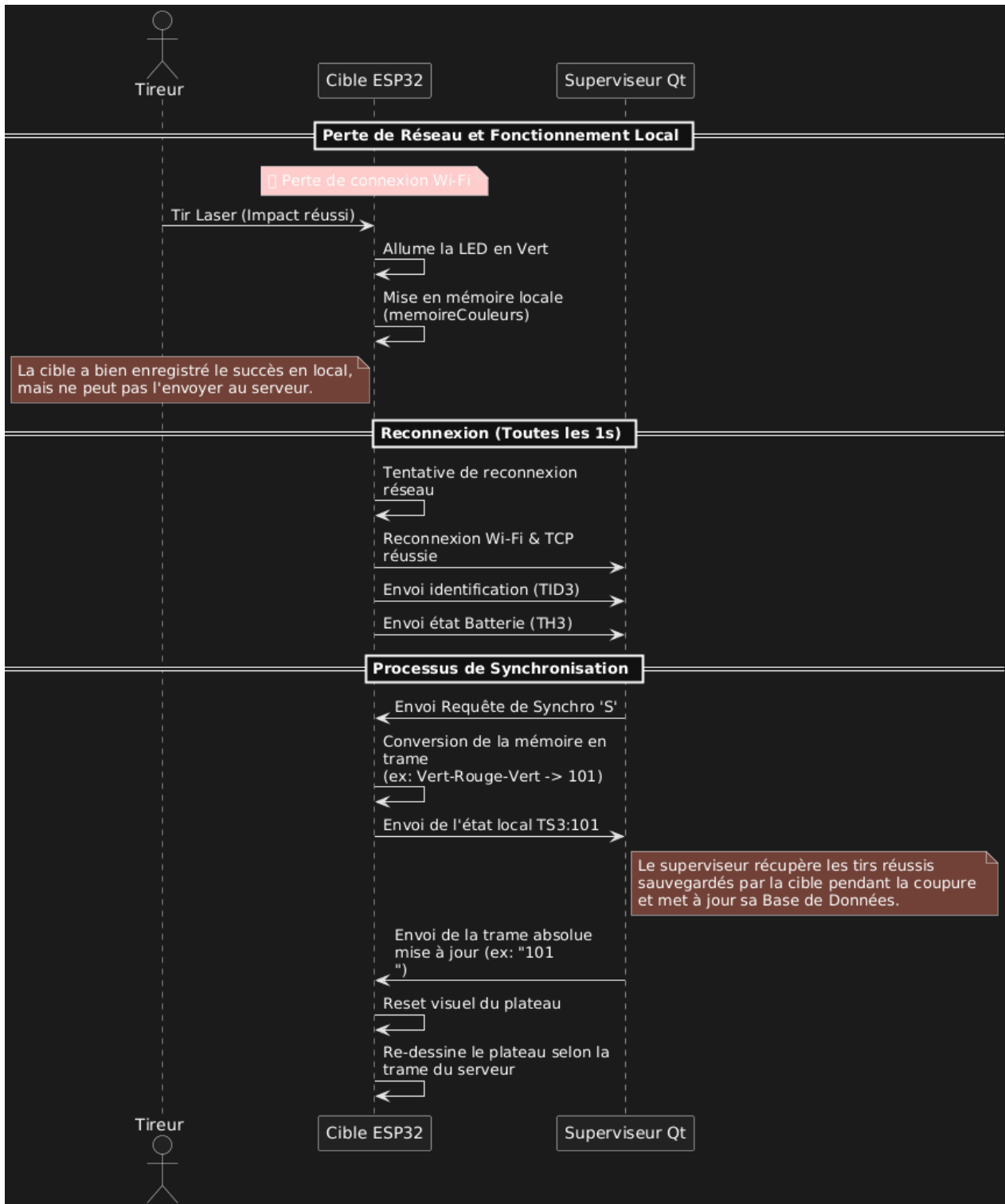
Enfin, l'intégralité des scénarios de tests a été validée avec succès, prouvant la capacité du système à maintenir l'intégrité de la base de données des scores face à des situations extrêmes comme des coupures réseau ou des baisses d'alimentation. Le système intègre également un monitoring de batterie efficace et une communication TCP désormais totalement sécurisée par un algorithme de chiffrement XOR, protégeant ainsi l'intégrité de la compétition contre le piratage. L'objectif d'allègement du programme principal a été mené à son terme avec une abstraction totale via la Programmation Orientée Objet (POO) et un fichier de configuration dédié. Le système est désormais capable d'être déployé et opérationnel pour les épreuves.

## 16. Annexes

### I. Diagramme de séquence du fonctionnement global de la cible



## II. Diagramme de séquence du fonctionnement de la synchronisation de la cible



# **Youssef El Allouchi**

*Partie Personnelle- Youssef El Allouchi*



|                                                                        |           |
|------------------------------------------------------------------------|-----------|
| <b>Lycée Alphonse Benoît .....</b>                                     | <b>1</b>  |
| Introduction .....                                                     | 3         |
| Diagramme de classes.....                                              | 4         |
| Diagramme de séquence .....                                            | 5         |
| 1. La base du serveur TCP et de la base de données .....               | 6         |
| 1.1. Un serveur qui gère plusieurs clients en même temps.....          | 6         |
| 1.2. La mémoire partagée (CZdc).....                                   | 7         |
| 2.Gérer les coupures de réseau (Anti-Déconnexion).....                 | 8         |
| 2.1. Sauvegarder les scores en direct avec des QMap .....              | 8         |
| 3.Piloter le matériel depuis le site Web.....                          | 8         |
| 3.1. Interroger la BDD avec un QTimer.....                             | 9         |
| 4.Sécuriser le code (Mutex et Anti-Crash) .....                        | 10        |
| 4.1. Empêcher les fils d'attente de se marcher dessus (Le Mutex) ..... | 10        |
| 4.2. Nettoyage mémoire et connexions SQL uniques .....                 | 11        |
| 5.La Synchro et Batteries .....                                        | 11        |
| 5.1. L'arbitrage automatique (Synchronisation).....                    | 11        |
| 5.2. Surveiller les batteries .....                                    | 12        |
| 6.Le wifi.....                                                         | 13        |
| Problème Wifi .....                                                    | 13        |
| <b>7.Site .....</b>                                                    | <b>14</b> |
| <b>8.SUJET DE PHYSIQUE .....</b>                                       | <b>17</b> |
| <b>9.CALENDRIER .....</b>                                              | <b>19</b> |

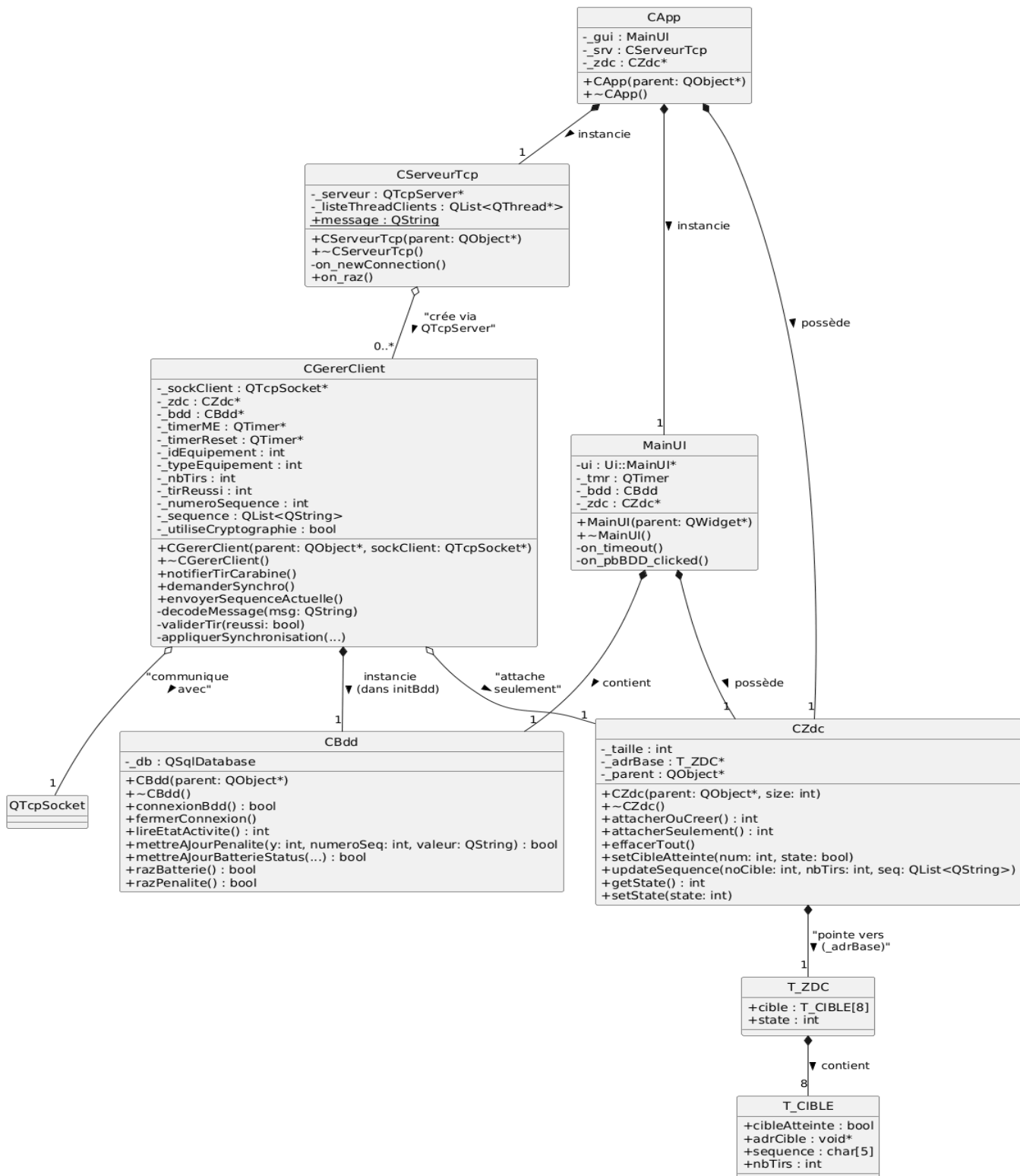
# Introduction

Le projet **Biathlon** a pour mission de numériser et de fiabiliser la détection de tirs laser sur des cibles physiques, tout en centralisant les données sur un superviseur logiciel développé sous **Qt (C++)**.

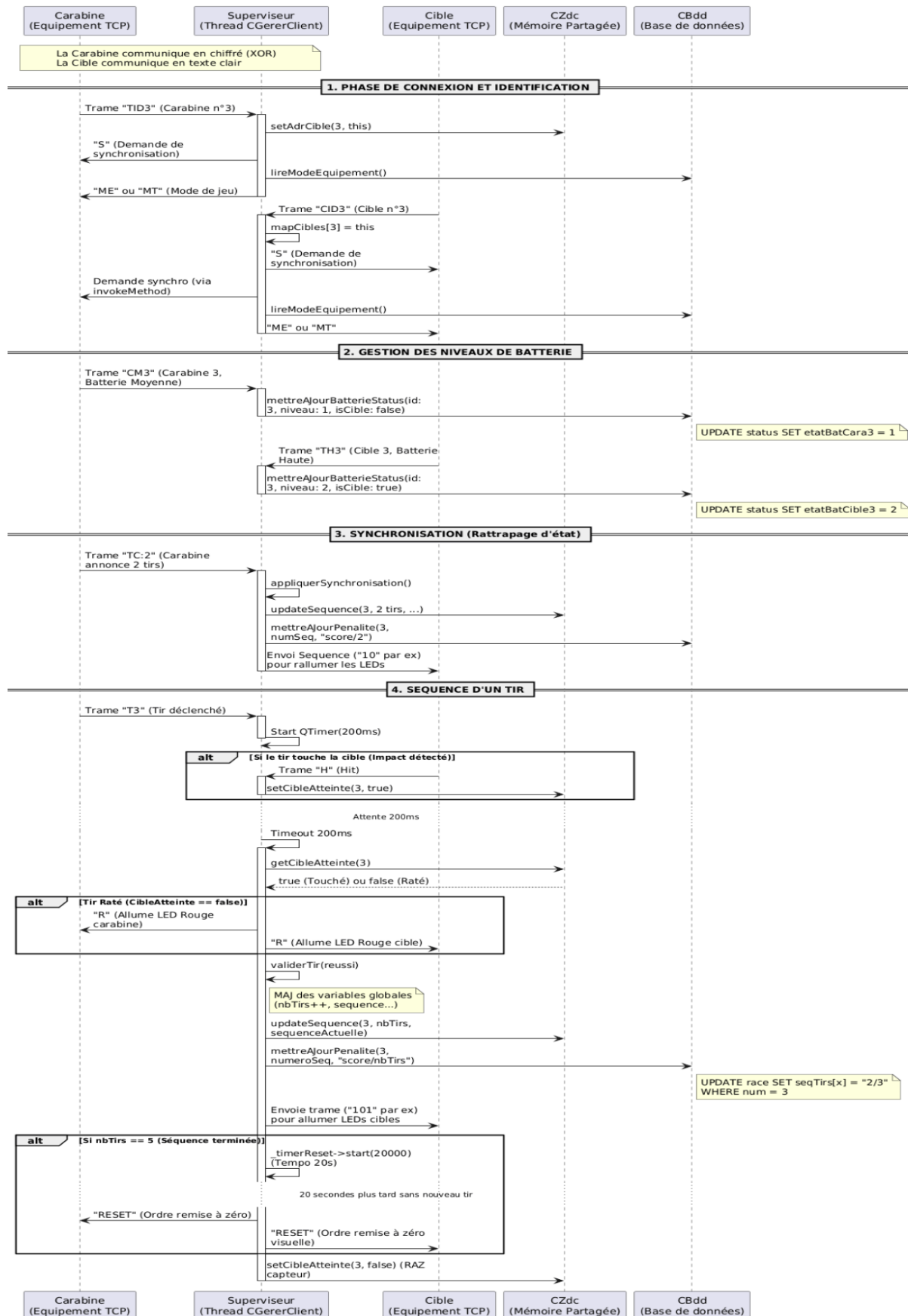
Au fil du développement de la version initiale, plusieurs limites ont été identifiées, impactant directement la fiabilité du système en conditions réelles : instabilités réseau lors de l'utilisation en extérieur, risques de corruption de données en environnement multithread et manque de visibilité sur l'état matériel des équipements.

L'objectif de cette dernière version est de proposer une refonte profonde de l'architecture pour transformer un prototype fonctionnel en un bon système de supervision, sécurisé et prêt pour l'exploitation. Cette montée en version s'articule autour de trois axes d'amélioration.

# Diagramme de classes



# Diagramme de séquence



# 1. La base du serveur TCP et de la base de données

Le plus gros défi ici, c'était de créer un lien solide entre le matériel (les cibles et carabines) et le programme de supervision, tout en commençant à enregistrer les scores.

## 1.1. Un serveur qui gère plusieurs clients en même temps

Au lieu de faire une simple boucle qui attend que chaque message se termine (ce qui bloquerait le serveur), j'ai choisi de faire un serveur TCP asynchrone avec QTcpServer. L'astuce technique se trouve dans la fonction `on_newConnection()`. Dès qu'un équipement se branche sur le port 2222, le serveur récupère le socket avec `nextPendingConnection()`, crée un objet `CGererClient`, et le place dans un nouveau `QThread`.

**En clair :** Ça permet au serveur de gérer plein d'équipements en même temps sans jamais ramer. Si une carabine met du temps à traiter une info, la cible d'à côté qui tourne sur un autre thread ne s'en rend même pas compte.

```
void CServeurTcp::on_newConnection()
{
 QTcpSocket *sockClient = _serveur->nextPendingConnection();

 CGererClient *client = new CGererClient(nullptr, sockClient);

 // CORRECTION CRITIQUE ANTI-CRASH : Le socket devient un enfant du client
 // S'il y a déconnexion, tout sera nettoyé en même temps sans plantage.
 sockClient->setParent(client);

 QThread *th = new QThread();
 client->moveToThread(th);

 // 1. Quand le socket se déconnecte, on demande au Thread de s'arrêter
 connect(sockClient, &QTcpSocket::disconnected, th, &QThread::quit);
 connect(sockClient, &QTcpSocket::disconnected, [](){
 qDebug() << "⚠ DECONNEXION : Un équipement a perdu le réseau. Nettoyage du Thread...";
 });

 // 2. Quand le Thread s'arrête, il détruit le client (qui détruit le socket)
 connect(th, &QThread::finished, client, &QObject::deleteLater);

 // 3. Le Thread se détruit lui-même
 connect(th, &QThread::finished, th, &QObject::deleteLater);

 connect(this, &CServeurTcp::sig_raz, client, &CGererClient::on_raz);
 connect(th, &QThread::started, client, &CGererClient::initBdd);

 _listeThreadClients.append(th);
 th->start();

 qDebug() << "✅ NOUVELLE CONNEXION : Equipement connecté depuis l'IP :" << sockClient->peerAddress().toString();
}

void CServeurTcp::on_raz()
```

## 1.2. La mémoire partagée (CZdc)

Comme chaque équipement tourne dans son propre thread isolé, il me fallait une solution pour qu'ils puissent échanger des infos (ex: savoir si une cible est touchée quand on valide un tir). J'ai créé la classe CZdc en utilisant QSharedMemory. En gros, c'est une petite RAM virtuelle avec une structure T\_ZDC contenant l'état des 5 cibles, où tous les threads peuvent lire et écrire en direct en utilisant des lock() et unlock().

méthode : updateSequence

```
void CZdc::updateSequence(int noCible, int nbTirs, QList<QString> seq)
{
 if (!_adrBase || noCible < 1 || noCible > 8) return;

 lock();
 _adrBase->cible[noCible-1].nbTirs = nbTirs;
 memset(_adrBase->cible[noCible-1].sequence, 0, sizeof(_adrBase->cible[noCible-1].sequence));

 for (int i=0 ; i<seq.size() && i<5 ; i++) {
 _adrBase->cible[noCible-1].sequence[i] = (seq.at(i)=="1"? '1': '0');
 }
 unlock();
}
```

méthode : setCibleAtteinte

```
void CZdc::setCibleAtteinte(int num, bool state)
{
 if (!_adrBase || num < 1 || num > 8) return;

 lock();
 _adrBase->cible[num-1].cibleAtteinte = state;
 unlock();
}
```

## 2.Gérer les coupures de réseau (Anti-Déconnexion)

En testant dehors, on a vu que le Wi-Fi des ESP32 pouvait sauter. Si ça coupait, on perdait tout le score du coureur. Il fallait une "mémoire globale".

### 2.1. Sauvegarder les scores en direct avec des QMap

Pour régler ça, j'ai déclaré des variables statiques tout en haut de cgererclient.cpp, comme QMap<int, int> memoireSequence et memoireNbTirs. C'est comme des dictionnaires qui lient l'ID du matériel à son score.

**En clair :** Quand un équipement se connecte, il envoie son ID (la trame TID:x pour une cible ou CID:x pour une carabine). Le serveur lit cette trame dans decodeMessage(), vérifie si cet ID est déjà dans la QMap, et si c'est le cas, il sait exactement où en était le coureur avant la coupure.

```
#include "cgererclient.h"
#include <QMap>
#include <QMutex>

// --- MÉMOIRE GLOBALE ---
static QMap<int, int> memoireSequence;
static QMap<int, int> memoireNbTirs;
static QMap<int, int> memoireTirReussi;
static QMap<int, QList<QString>> memoireListeSequence;
static QMutex memoireMutex; // SÉCURITÉ THREADS
// -----
```

## 3.Piloter le matériel depuis le site Web

je voulais que le site web puisse commander les équipements, par exemple pour passer du mode "Entraînement" (MT) au mode "Examen" (ME) sans toucher au matériel physique.

Mode Entraînement

Mode Examen

### 3.1. Interroger la BDD avec un QTimer

Dans cgererclient.cpp, j'ai configuré un QTimer (\_timerME) qui tourne toutes les 10 secondes. À chaque fois, il appelle la fonction envoyerModeEquipement(). Le serveur demande à la base de données quel est le mode actuel (via \_bdd->lireModeEquipement()). Si le mode a changé sur le site web, le serveur envoie directement la trame "ME" ou "MT" dans le socket TCP pour mettre à jour l'ESP32.

```
void CGererClient::envoyerModeEquipement()
{
 if (_bdd && _sockClient && _sockClient->state() == QAbstractSocket::ConnectedState) {
 int modeME = _bdd->lireModeEquipement();
 if (modeME == 1) {
 qDebug() << "🔵 [ENVOI] Mode Examen (ME) envoyé à l'équipement" << _idEquipement;
 _sockClient->write("ME\n");
 } else if (modeME == 0) {
 qDebug() << "🔵 [ENVOI] Mode Entraînement (MT) envoyé à l'équipement" << _idEquipement;
 _sockClient->write("MT\n");
 }
 _sockClient->flush();
 }
}
```

## 4. Sécuriser le code (Mutex et Anti-Crash)

Plus on ajoutait de choses, plus on risquait des bugs liés au fait que plusieurs threads

### 4.1. Empêcher les fils d'attente de se marcher dessus (Le Mutex)

Si la carabine et la cible envoyaient une info exactement au même moment, deux threads essaient de modifier la variable `memoireNbTirs` en même temps. Ça corrompait les données (Race Condition). J'ai ajouté un objet `QMutex` `memoireMutex`. Maintenant, avant de modifier un score dans `validerTir()`, le thread fait un `memoireMutex.lock()`. Il fait son calcul tranquillement, puis fait un `unlock()`. Les autres doivent attendre leur tour.

```
void CGenerClient::validerTir(bool reussi)
{
 int state = _zdc->getState();
 if (state != 3) {
 qDebug() << " [BLOCAGE] Tir ignoré car la course n'est pas lancée.";
 return;
 }

 memoireMutex.lock();
 _nbTirs = memoireNbTirs.value(_idEquipement, 0);
 _numeroSequence = memoireSequence.value(_idEquipement, 1);
 _sequence = memoireListeSequence.value(_idEquipement, QList<QString>());
 _tirReussi = memoireTirReussi.value(_idEquipement, 0);
 memoireMutex.unlock();

 _nbTirs++;

 if (_nbTirs == 6) {
 _nbTirs = 1;
 _tirReussi = 0;
 _numeroSequence++;
 _sequence.clear();
 if (_numeroSequence == 4) _numeroSequence = 1;
 }

 _sequence.append(reussi ? "1" : "0");
 if (reussi) _tirReussi++;

 qDebug() << "\n===== ";
 qDebug() << " TIR DÉTECTÉ - EQUIPEMENT N°" << _idEquipement;
 qDebug() << " Résultat : " << (reussi ? " TOUCHÉ" : " RATÉ");
 qDebug() << " Score de la seq:" << _tirReussi << "/" << _nbTirs;
 qDebug() << "===== \n";

 memoireMutex.lock();
 memoireSequence[_idEquipement] = _numeroSequence;
 memoireNbTirs[_idEquipement] = _nbTirs;
 memoireTirReussi[_idEquipement] = _tirReussi;
 memoireListeSequence[_idEquipement] = _sequence;
 memoireMutex.unlock();

 _zdc->updateSequence(_idEquipement, _nbTirs, _sequence);

 if (_bdd) {
 QString trameBdd = QString::number(_tirReussi) + "/" + QString::number(_nbTirs);
 _bdd->mettreAJourPenalite(_idEquipement, _numeroSequence, trameBdd);
 }
}
```

## 4.2. Nettoyage mémoire et connexions SQL uniques

Pour éviter les crashes à la déconnexion, j'ai lié le socket à son client (`sockClient->setParent(client)`) et j'utilise les signaux pour faire un `deleteLater()` propre quand le thread se termine. De plus, pour éviter que les threads se mélangent dans la base de données, chaque thread crée sa propre connexion SQL avec un nom unique grâce à `QThread::currentThreadId()` dans `cbdd.cpp`.

```
#include "cbdd.h"

CBdd::CBdd(QObject *parent)
 : QObject(parent)
{
 QString connName = QString("mysql_thread_%1").arg(reinterpret_cast<quintptr>(QThread::currentThreadId()));
 _db = QSqlDatabase::addDatabase("QMYSQL", connName);
 _db.setHostName("localhost");
 _db.setDatabaseName("biathlon");
 _db.setUserName("biathlon");
 _db.setPassword("admin");
}

CBdd::~CBdd()
{
 QString connName = _db.connectionName();
 fermerConnexion();
 _db = QSqlDatabase();
 QSqlDatabase::removeDatabase(connName);
}
```

## 5. La Synchro et Batteries

### 5.1. L'arbitrage automatique (Synchronisation)

J'ai codée la méthode `appliquerSynchronisation()`. Si la carabine envoie la trame "TC:5" (elle pense avoir tiré 5 fois) mais que la cible envoie "TS:1100" (elle n'a vu que 4 tirs), le serveur croise les listes, prend la valeur la plus haute, met à jour sa propre mémoire, et renvoie la bonne séquence pour forcer les deux équipements à se mettre d'accord.

```

void CGenerClient::appliquerSynchronisation(int tirsEquipement, QString seqEquipement, bool vientDeCarabine)
{
 qDebug() << " [SYNCHRO EN COURS] Equipement:" << _idEquipement;
 qDebug() << " - Données reçues -> Vient de carabine:" << vientDeCarabine << " | Tirs carabine:" << tirsEquipement << " | Séquence cible

 memoireMutex.lock();
 int tirsSup = memoireNbTirs.value(_idEquipement, 0);
 QList<QString> listeSeqSup = memoireListeSequence.value(_idEquipement, QList<QString>());
 int numSeq = memoireSequence.value(_idEquipement, 1);
 memoireMutex.unlock();

 if (vientDeCarabine) {
 if (tirsEquipement > tirsSup) tirsSup = tirsEquipement;
 } else {
 listeSeqSup.clear();
 for(QChar c : seqEquipement) {
 if (c == '0' || c == '1') listeSeqSup.append(QString(c));
 }
 }

 int missing = tirsSup - listeSeqSup.size();
 for (int i = 0; i < missing; i++) listeSeqSup.append("0");

 int tirReussi = 0;
 for(const QString &val : listeSeqSup) {
 if (val == "1") tirReussi++;
 }

 memoireMutex.lock();
 memoireNbTirs[_idEquipement] = tirsSup;
 memoireListeSequence[_idEquipement] = listeSeqSup;
 memoireTirReussi[_idEquipement] = tirReussi;
 memoireMutex.unlock();

 _nbTirs = tirsSup;
 _sequence = listeSeqSup;
 _tirReussi = tirReussi;

 _zdc->updateSequence(_idEquipement, tirsSup, listeSeqSup);

 if (_bdd) {
 QString trameBdd = QString::number(tirReussi) + "/" + QString::number(tirsSup);
 _bdd->mettreAJourPenalite(_idEquipement, numSeq, trameBdd);
 }

 CGenerClient *cible = (CGenerClient *)_zdc->lireAdrCible(_idEquipement);
 if (cible) {

```

## 5.2. Surveiller les batteries

Le serveur sait maintenant lire les trames de batterie : TH/TM/TL pour la cible et CH/CM/CL pour la carabine. Quand decodeMessage() voit ça, il appelle directement mettreAJourBatterieStatus() pour mettre la jauge à jour sur le site web.

```

 int niveau = 0;
 if (msg.startsWith("TH")) niveau = 2;
 else if (msg.startsWith("TM")) niveau = 1;
 else if (msg.startsWith("TL")) niveau = 0;

 int idEq = msg.mid(2).toInt(); // Récupère l'ID
 if (_bdd) _bdd->mettreAJourBatterieStatus(idEq, niveau, true);
}

// GESTION BATTERIE CARABINE/LASER (CHx, CMx, CLx)
else if (msg.startsWith("CH") || msg.startsWith("CM") || msg.startsWith("CL")) {
 int niveau = 0;
 if (msg.startsWith("CH")) niveau = 2;
 else if (msg.startsWith("CM")) niveau = 1;
 else if (msg.startsWith("CL")) niveau = 0;

```

## 6.Le wifi

L'étude de l'architecture, la configuration de l'environnement réseau (mise en place du Raspberry Pi en point d'accès WiFi autonome) et l'installation de l'environnement de développement complet (Qt Creator et MariaDB) ont été réalisées dans les temps. J'ai également finalisé la structure des données (importation du fichier `biathlon.sql` et liaison réussie avec l'application) ainsi que le lancement opérationnel du serveur TCP sur le port 2222, prêt à recevoir les connexions.

### Problème Wifi

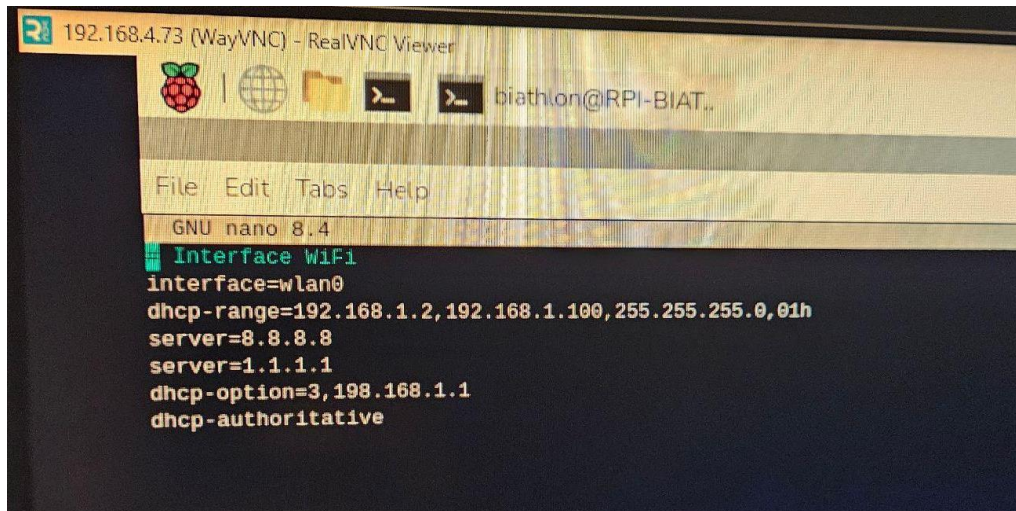
**problème :** Le réseau WiFi (Hotspot) fonctionnait correctement juste après sa création avec le logiciel automatisé RaspAP, mais il n'était plus diffusé ou devenait totalement inaccessible à chaque redémarrage du système

**Cause :** L'interface automatisée RaspAP créait des conflits avec les gestionnaires de réseau natifs des versions récentes de Raspberry Pi OS (conflit entre NetworkManager et dhcpcd). Les processus de démarrage s'annulaient entre eux, empêchant le réseau de s'activer de manière autonome.

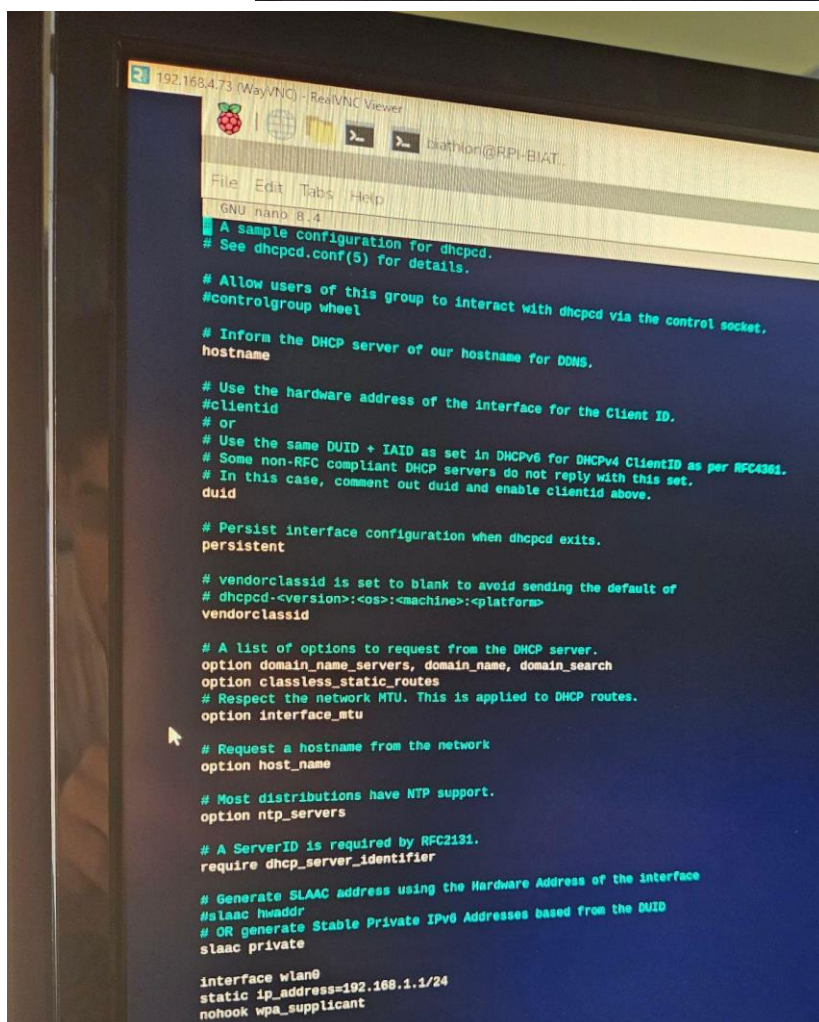
**Solution :** Désinstallation de la surcouche RaspAP au profit d'une configuration réseau manuelle, native et plus stable. J'ai paramétré indépendamment les services de base : `hostapd` pour la gestion de l'antenne radio (création du réseau BIATHLON), `dnsmasq` pour le serveur DHCP (distribution des adresses IP aux cibles), et la modification du fichier `/etc/dhcpcd.conf` pour fixer l'adresse IP statique du superviseur.

Pour simplifier nos déploiements sur le terrain ,j'ai décidé de rendre notre Raspberry Pi totalement autonome. Je me suis appuyé sur **NetworkManager** (l'outil de gestion réseau par défaut sous Linux, qui est vraiment robuste). Je l'ai configuré pour que la carte Wi-Fi de la Raspberry se comporte comme un point d'accès (hotspot).

J'ai nommé ce réseau "**Biathlon**" et je l'ai bien sûr sécurisé avec un mot de passe. Le gros avantage, c'est que maintenant, dès qu'on branche la Raspberry, elle crée son propre réseau Wi-Fi. Nos carabines et nos cibles s'y connectent automatiquement et commencent à discuter avec le serveur TCP sans qu'on ait besoin de configurer quoi que ce soit sur place.



```
192.168.4.73 (WayVNC) - RealVNC Viewer
biathlon@RPI-BIATL:~$ nano /etc/dhcp/dhclient.conf
GNU nano 8.4
Interface WiFi
interface=wlan0
dhcp-range=192.168.1.2,192.168.1.100,255.255.255.0,01h
server=8.8.8.8
server=1.1.1.1
dhcp-option=3,198.168.1.1
dhcp-authoritative
```



```
192.168.4.73 (WayVNC) - RealVNC Viewer
biathlon@RPI-BIATL:~$ nano /etc/dhcp/dhclient.conf
GNU nano 8.4
A sample configuration for dhclient.
See dhclient.conf(5) for details.

Allow users of this group to interact with dhclient via the control socket.
#controlgroup wheel

Inform the DHCP server of our hostname for DNS.
hostname

Use the hardware address of the interface for the Client ID.
#clientid
or
Use the same DUID + IAID as set in DHCPv6 for DHCPv4 ClientID as per RFC4361.
Some non-RFC compliant DHCP servers do not reply with this set.
In this case, comment out duid and enable clientid above.
duid

Persist interface configuration when dhclient exits.
persistent

vendorclassid is set to blank to avoid sending the default of
dhclient-<version>:<os>:<machine>:<platform>
vendorclassid

A list of options to request from the DHCP server.
option domain_name_servers, domain_name, domain_search
option classless_static_routes
Respect the network MTU. This is applied to DHCP routes.
option interface_mtu

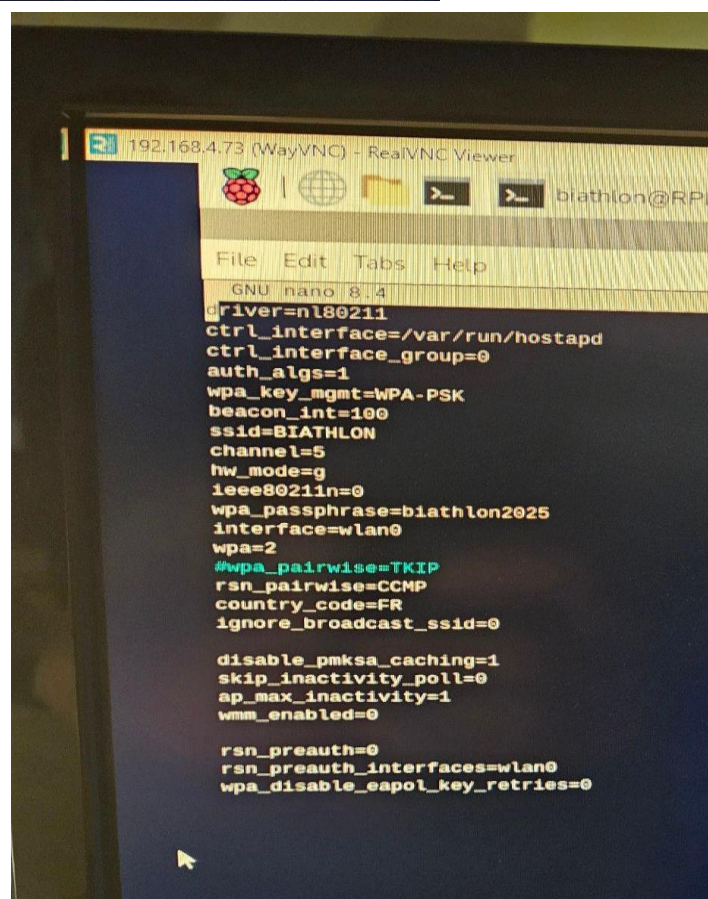
Request a hostname from the network
option host_name

Most distributions have NTP support.
option ntp_servers

A ServerID is required by RFC2131.
require dhcp_server_identifier

Generate SLAAC address using the Hardware Address of the interface
#slaac hwaddr
OR generate Stable Private IPv6 Addresses based from the DUID
slaac private

interface wlan0
static ip_address=192.168.1.1/24
nohook wpa_supplicant
```



```
192.168.4.73 (WayVNC) - RealVNC Viewer
biathlon@RPI-BIATL:~$ nano /etc/wpa_supplicant/wpa_supplicant.conf
GNU nano 8.4
driver=nl80211
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
auth_algs=1
wpa_key_mgmt=WPA-PSK
beacon_int=100
ssid=BIATHLON
channel=5
hw_mode=g
ieee80211n=0
wpa_passphrase=biathlon2025
interface=wlan0
wpa=2
#wpa_pairwise=TKIP
rsn_pairwise=CCMP
country_code=FR
ignore_broadcast_ssid=0

disable_pmksa_caching=1
skip_inactivity_poll=0
ap_max_inactivity=1
wmm_enabled=0

rsn_preauth=0
rsn_preauth_interfaces=wlan0
wpa_disable_eapol_key_retries=0
```

## 7.Site

Concernant le site web, je l'ai tout d'abord sécurisé en le passant en HTTPS afin de prévenir toute faille de sécurité. Ensuite, je présente ci-dessous ses trois différentes interfaces :

- **L'interface Master** (destinée à l'arbitre ou au professeur) ;
- **L'interface Juge** (destinée à l'élève évaluateur) ;
- **L'interface Public** (destinée aux autres élèves / spectateurs).

### BIATHLON PARAMETRES MASTER

Nom course :

Nombre de juges :

Distance d'un tours en m (1T) :

Distance 2 tours en m (2T) :

Distance SANS pénalité en m (TP) :

Distance du tour de pénalité en m (TP) :

[Changer le code](#)

[RAZ](#)

[Définissez les paramètres...](#)

### Fonctionnement :

Tout d'abord, l'arbitre définit le nom de la course et le nombre de juges. Une fois cette étape validée, si l'arbitre a choisi 1 juge (par exemple), l'arbitre a terminé sa tâche pour cette partie. L'état s'incrémente alors de 1 dans la base de données (l'état initial étant à 0), et l'arbitre passe en attente des juges.

### BIATHLON PARAMETRES JUGE

Votre nom (juge) :

Nom du coureur :

[Définissez les paramètres...](#)

Ensuite, le juge saisit son nom ainsi que celui du coureur qu'il va évaluer. Une fois cette étape terminée, il passe en attente du lancement de la course. L'état s'incrémente à nouveau de 1, passant ainsi à 2.

## BIATHLON LANCEMENT COURSE MASTER

Nom de la course : course 1  
Nombre de coureurs/juges : 1

-----  
Youssef est le juge 1 de Thomas

GO [RAZ](#)

Cliquez pour démarrer la course !

Enfin, l'arbitre lance la course en cliquant sur « GO ». L'état s'incrémente une nouvelle fois pour passer à 3, et la course démarre alors en mode examen.

### Course en cours !

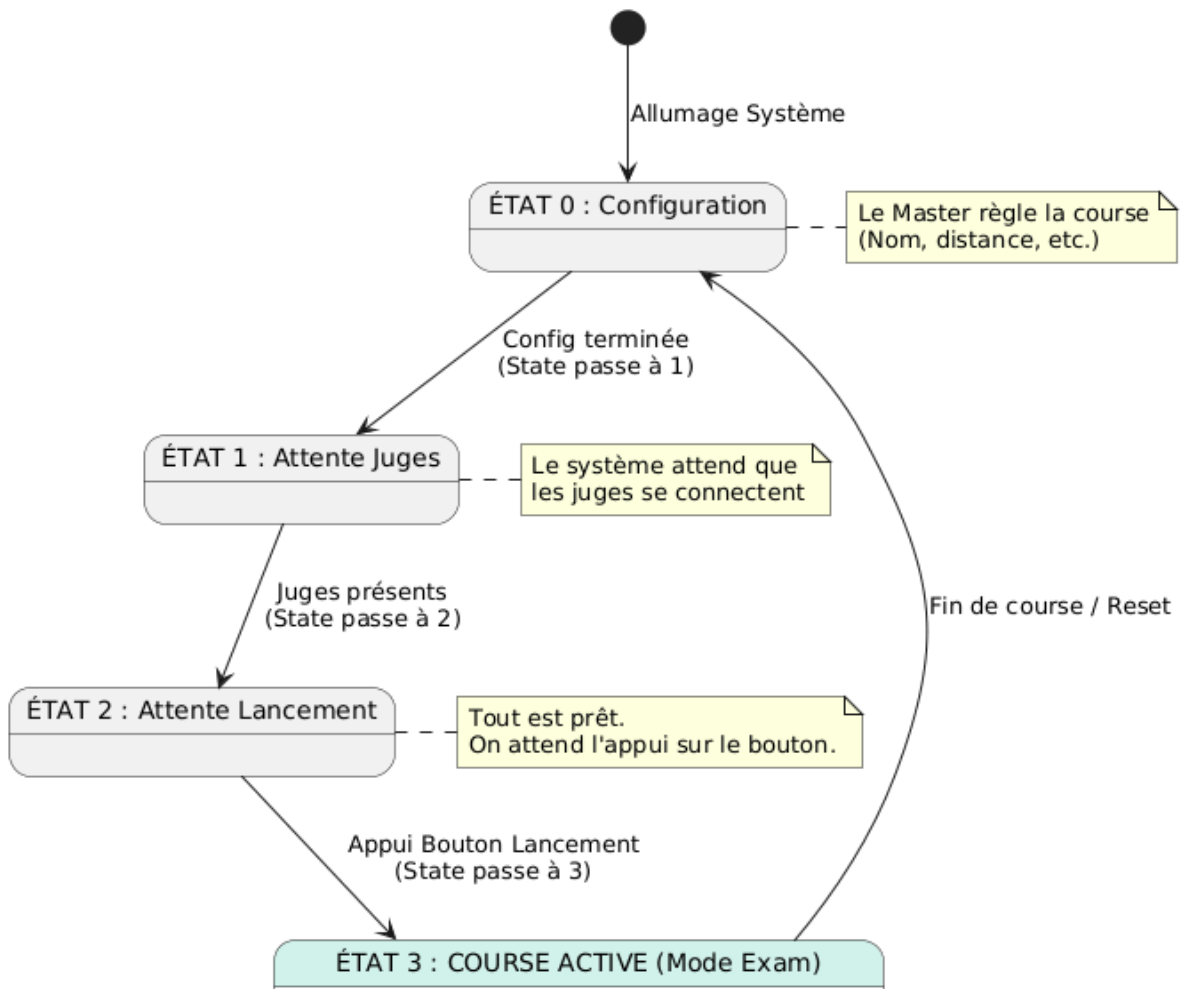
| Num | Coureur | 1T1 | ST1 | TP1 | 2T2 | ST2 | TP2 | 2T3 | ST3 | TP3 | 1T2 | Total |
|-----|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------|
| 1   | Thomas  |     |     |     |     |     |     |     |     |     |     |       |

Temps au format min:sec:cent. 2T : 2 tours de stade. ST : Séquence de 5 tirs. TP : Tours de pénalité.

Enfin, les spectateurs qui le souhaitent ont la possibilité de suivre la course en temps réel depuis leur propre interface Public.

# Diagramme d'état

Cycle de l'Épreuve Biathlon - États du Superviseur



## 8.SUJET DE PHYSIQUE

Je voulais vous présenter une réflexion que j'ai eue récemment en travaillant sur mon projet avec la Raspberry Pi.

En me penchant sur les problèmes de connexion de ma carte, je me suis rappelé que ce n'est rien d'autre que de la lumière invisible : des ondes électromagnétiques.

**1. Une observation de terrain : 2.4 GHz contre 5 GHz** Pendant les phases de test, j'ai remarqué un comportement qui m'a d'abord paru super bizarre, mais qui est en fait totalement logique. Ma Raspberry Pi possède une petite antenne pour capter le signal de ma box internet. L'interface me laisse le choix entre deux bandes de fréquences :

le 2.4 GHz et le 5 GHz, Voici ce que j'ai constaté :

- Quand je configure ma Raspberry Pi sur le réseau **2.4 GHz**, je peux aller dans une autre pièce, fermer la porte, et mon Pi capte toujours le signal à fond.
- À l'inverse, quand je passe en **5 GHz** (qui me donne des débits informatiques tellement plus rapides, parfaits pour télécharger mes paquets de code), le signal s'effondre dès que je change de pièce et qu'un simple mur se dresse entre la box et la carte.

**2. La preuve par la physique** Je me suis replongé dans mon cours de physique pour comprendre ce phénomène. Il y a une règle concernant les ondes électromagnétiques : la longueur d'une onde est inversement proportionnelle à sa fréquence. Plus la fréquence est élevée, plus l'onde est petite. Plus la fréquence est basse, plus l'onde est longue et ample.

J'ai voulu modéliser cela concrètement avec l'équation de propagation :  $\lambda = c / f$  (où  $\lambda$  est la longueur d'onde,  $c$  la vitesse de la lumière soit environ 300 000 000 m/s, et  $f$  la fréquence en Hertz), Voici le calcul exact pour mon réseau 2.4 GHz :

- $c = 3 \times 10^8$  m/s
- $f = 2,4 \times 10^9$  Hz (2,4 GHz)
- $\lambda = (3 \times 10^8) / (2,4 \times 10^9) = 0,125$  mètre, soit **12,5 centimètres**.

Et pour le 5 GHz, la fréquence étant environ deux fois plus élevée, la longueur d'onde est divisée par deux, tombant à **environ 6 cm**.

**3. Ce que ça change concrètement pour mon projet** Ce chiffre est de 12,5 centimètres.

- **En 2.4 GHz (12,5 cm) :** Les ondes sont assez "épaisses" et amples pour traverser des murs ou se diffracter facilement autour des obstacles. C'est pour ça que ma Raspberry Pi capte bien de loin.
- **En 5 GHz (6 cm) :** Ces petites ondes transportent beaucoup plus de données à la seconde, mais elles sont trop courtes. Face aux murs, elles peinent à passer et se font massivement absorber ou bloquer.

**Bilan :** J'ai donc décidé d'adapter la configuration réseau à l'usage physique de mon projet.

# 9.CALENDRIER

## Prévisionnel

|                    |    | physique                                                 |       | 04/03/26     | 06/03/26     |    |
|--------------------|----|----------------------------------------------------------|-------|--------------|--------------|----|
| DIAGRAMME DE GANTT | 30 | 4 activité IR 3                                          | 158 h | Lun 05/01/26 | Mar 05/05/26 |    |
|                    | 31 | Lecture du dossier Biathlon                              | 12 h  | Lun 05/01/26 | Lun 12/01/26 |    |
|                    | 32 | microsoft project                                        | 8 h   | Lun 12/01/26 | Mer 14/01/26 | 31 |
|                    | 33 | Mise en place du wifi                                    | 20 h  | Mer 14/01/26 | Mar 27/01/26 | 32 |
|                    | 34 | amélioration programme de réception du fusil             | 24 h  | Mar 27/01/26 | Mar 10/02/26 | 33 |
|                    | 35 | réglage de la base de donnée                             | 12 h  | Mar 10/02/26 | Mar 17/02/26 | 34 |
|                    | 36 | réglage du serveur TCP                                   | 24 h  | Mar 17/02/26 | Mar 17/03/26 | 35 |
|                    | 37 | réglage et amélioration des différent problème rencontré | 20 h  | Mar 17/03/26 | Lun 30/03/26 | 36 |
|                    | 38 | sujet de physique                                        | 8 h   | Lun 30/03/26 | Mer 01/04/26 | 37 |
|                    | 39 | communication (BDD + fusil + cible)                      | 30 h  | Mer 01/04/26 | Mar 05/05/26 | 38 |

## Réel

|    |  | physique                                                 |       | 04/03/26     | 06/03/26     |    |
|----|--|----------------------------------------------------------|-------|--------------|--------------|----|
| 30 |  | 4 activité IR 3                                          | 180 h | Lun 05/01/26 | Mar 19/05/26 |    |
| 31 |  | Lecture du dossier Biathlon                              | 12 h  | Lun 05/01/26 | Lun 12/01/26 |    |
| 32 |  | microsoft project                                        | 8 h   | Lun 12/01/26 | Mer 14/01/26 | 31 |
| 33 |  | Mise en place du wifi                                    | 4 h   | Mer 14/01/26 | Lun 19/01/26 | 32 |
| 34 |  | amélioration programme de réception du fusil             | 20 h  | Lun 19/01/26 | Mer 28/01/26 | 33 |
| 35 |  | communication (BDD + fusil + cible)                      | 70 h  | Mer 28/01/26 | Mar 24/03/26 | 34 |
| 36 |  | réglage du serveur TCP                                   | 4 h   | Mar 24/03/26 | Mer 25/03/26 | 35 |
| 37 |  | réglage et amélioration des différent problème rencontré | 24 h  | Mer 25/03/26 | Mer 08/04/26 | 36 |
| 38 |  | sujet de physique                                        | 8 h   | Mer 08/04/26 | Mer 15/04/26 | 37 |
| 39 |  | réglage de la base de donnée                             | 30 h  | Mer 15/04/26 | Mar 19/05/26 | 38 |

## FICHE RECETTE, QUALIFICATION DU SYSTÈME

|                  |                      |                               |            |
|------------------|----------------------|-------------------------------|------------|
| Nom du système : | BIATHLON             | Code de la campagne de test : |            |
| Technicien 1 :   | Spinosi-baumet Lucas | Date :                        | 30/05/2026 |
| Technicien 2 :   | Retamosa thomas      |                               |            |
| Technicien 3 :   | El Allouchi Youssef  |                               |            |
| Technicien 4 :   | Klipel Timéo         |                               |            |
| Technicien 5 :   | Parisis Alexandre    |                               |            |

### IDENTIFICATION DU SCENARIO

| Identification du scénario de recette |                                                                                                                                                                              |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Titre :                               | Détection, validation d'un tir réussi et synchronisation réseau/BDD                                                                                                          |
| Objectif du scénario :                | Vérifier que la réception d'une trame de tir ("T3") suivie d'un impact ("H") incrémente le score, met à jour la base de données et ordonne l'allumage des LEDs sur la cible. |

### CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

- Le serveur logiciel QbiathlonV6 est démarré et écoute sur le port 2222
- La base de données MySQL locale (biathlon) est accessible et l'état de la course (state) vaut 3 (Course lancée).
- La Carabine est connecté (trame d'identification TIDX envoyée).
- La Cible est connecté (trame d'identification CIDX envoyée).

### EXÉCUTION DU TEST

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description courte : | 1 - Déclenchement du tir : La carabine envoie la trame (chiffée en XOR).<br>2 - Détection de l'impact : Dans le délai de 200 ms, la cible envoie la trame "H" (Hit).<br>3 - Validation (Fin temporisation) : Le timer de 200 ms expire.<br>4 - Retour Visuel : Le serveur notifie l'équipement physique de la cible.<br>5 - Persistance BDD : Vérification de l'enregistrement du score dans MySQL.                                                                                                                                                                                                            |
| Résultats attendus : | 1 - Le superviseur décrypte "Tx", identifie la cible visée et lance une temporisation de 200 ms.<br>2 - Le superviseur reçoit "H" et passe immédiatement la variable cibleAtteinte à true dans la mémoire partagée (CZdc).<br>3 - Le serveur lit cibleAtteinte == true, appelle validerTir(true) et affiche dans la console : "TIR DÉTECTÉ - ÉQUIPEMENT - TOUCHÉ".<br>4 - Une trame contenant la nouvelle séquence (ex: "1") est envoyée en clair via le socket de la cible.<br>5 - Une requête UPDATE a bien été exécutée sur la table race. Le champ seqTirs1 contient désormais la valeur "1/1" avec l'ID x |

### BILAN

Le système a parfaitement validé le tir, le score en base est correct et la LED s'est allumée instantanément.

### REMARQUES

Penser à vérifier le comportement si le tir arrive alors que l'état de la course n'est pas à 3, le tir doit être ignoré comme spécifié dans le code de validerTir.

### CONCLUSION

|        |          |            |  |
|--------|----------|------------|--|
| VALIDE | <b>X</b> | NON VALIDE |  |
|--------|----------|------------|--|

(Mettre une croix dans la case correspondante)

# **Thomas Retamosa**

*Partie Personnelle- Thomas Retamosa*



# TABLE DES MATIÈRES

|                                                                             |           |
|-----------------------------------------------------------------------------|-----------|
| 1. INTRODUCTION ET OBJECTIFS DU PROJET .....                                | 2         |
| 1.1 Contexte .....                                                          | 2         |
| 1.2 Objectifs techniques .....                                              | 3         |
| 2. ARCHITECTURE MATÉRIELLE ET CONFIGURATION .....                           | 3         |
| 2.1 Le Microcontrôleur ESP32 .....                                          | 3         |
| 2.2 Diagramme de classes du code .....                                      | 3         |
| 2.3 Diagramme de séquence .....                                             | 4         |
| 2.4 Configuration des entrées/sorties (GPIO) .....                          | 5         |
| 3. ÉTUDE PHYSIQUE : ACQUISITION DE LA TENSION BATTERIE .....                | 6         |
| 3.1 Problématique matérielle .....                                          | 6         |
| 3.2 Le Pont Diviseur de Tension et de Protection .....                      | 6         |
| 3.3 Le Convertisseur Analogique-Numérique (CAN) .....                       | 7         |
| 3.4 Synthèse et Traitement Logiciel .....                                   | 8         |
| 4. GESTION DE LA COMMUNICATION RÉSEAU .....                                 | 9         |
| 4.1 Connexion TCP et Wi-Fi .....                                            | 9         |
| 4.2 Protocole de messagerie (Trames) .....                                  | 10        |
| <b>5. Cybersécurité : Chiffrement et déchiffrement des trames TCP .....</b> | <b>12</b> |
| 6. LOGIQUE MÉTIER ET MODES DE FONCTIONNEMENT .....                          | 14        |
| 6.1 Gestion des actions physiques (Anti-rebond et États) .....              | 14        |
| 6.2 Mode Entraînement .....                                                 | 14        |
| 6.3 Mode Examen (Compétition) .....                                         | 15        |
| <b>7. Robustesse du réseau : Le système de synchronisation .....</b>        | <b>17</b> |
| 8. CALENDRIER .....                                                         | 18        |
| 9. CONCLUSION .....                                                         | 19        |

# 1. INTRODUCTION ET OBJECTIFS DU PROJET

## 1.1 Contexte

Le projet Biathlon s'inscrit dans le cadre du développement d'un système de tir sportif électronique. L'objectif est de remplacer les munitions réelles par un système laser sécurisé, tout en maintenant les sensations et les contraintes du tir de compétition. Le cœur du système repose sur un microcontrôleur ESP32. Dans cette revue j'aborderai seulement le code qui nous a servis de test avec la visée allumée pour pouvoir viser afin d'effectuer le plus de tests possibles et précis.

## 1.2 Objectifs techniques

L'enjeu principal est de créer une interface fiable entre le matériel (la carabine) et un logiciel de supervision centralisé. Le système doit :

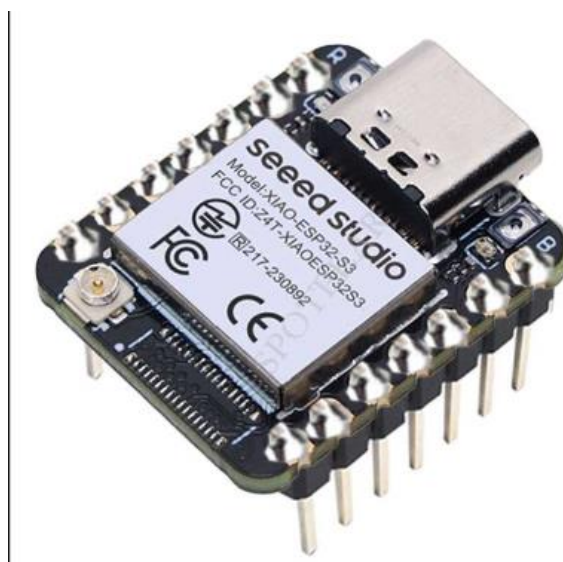
- Gérer deux modes de fonctionnement distincts : "Entraînement" (tirs illimités) et "Examen" (règles strictes de compétition).
- Assurer une communication TCP/IP robuste pour transmettre chaque tir en temps réel.

---

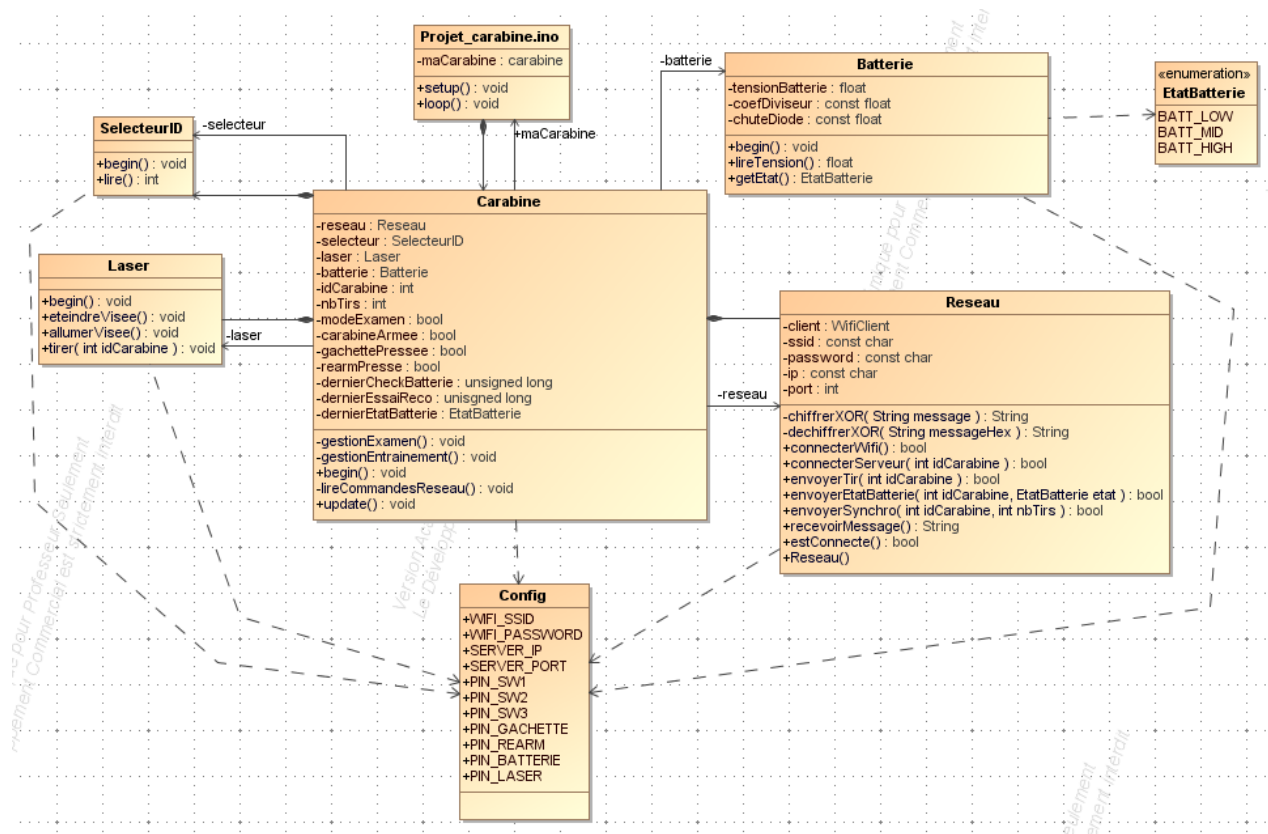
## 2. ARCHITECTURE MATÉRIELLE ET CONFIGURATION

### 2.1 Le Microcontrôleur ESP32

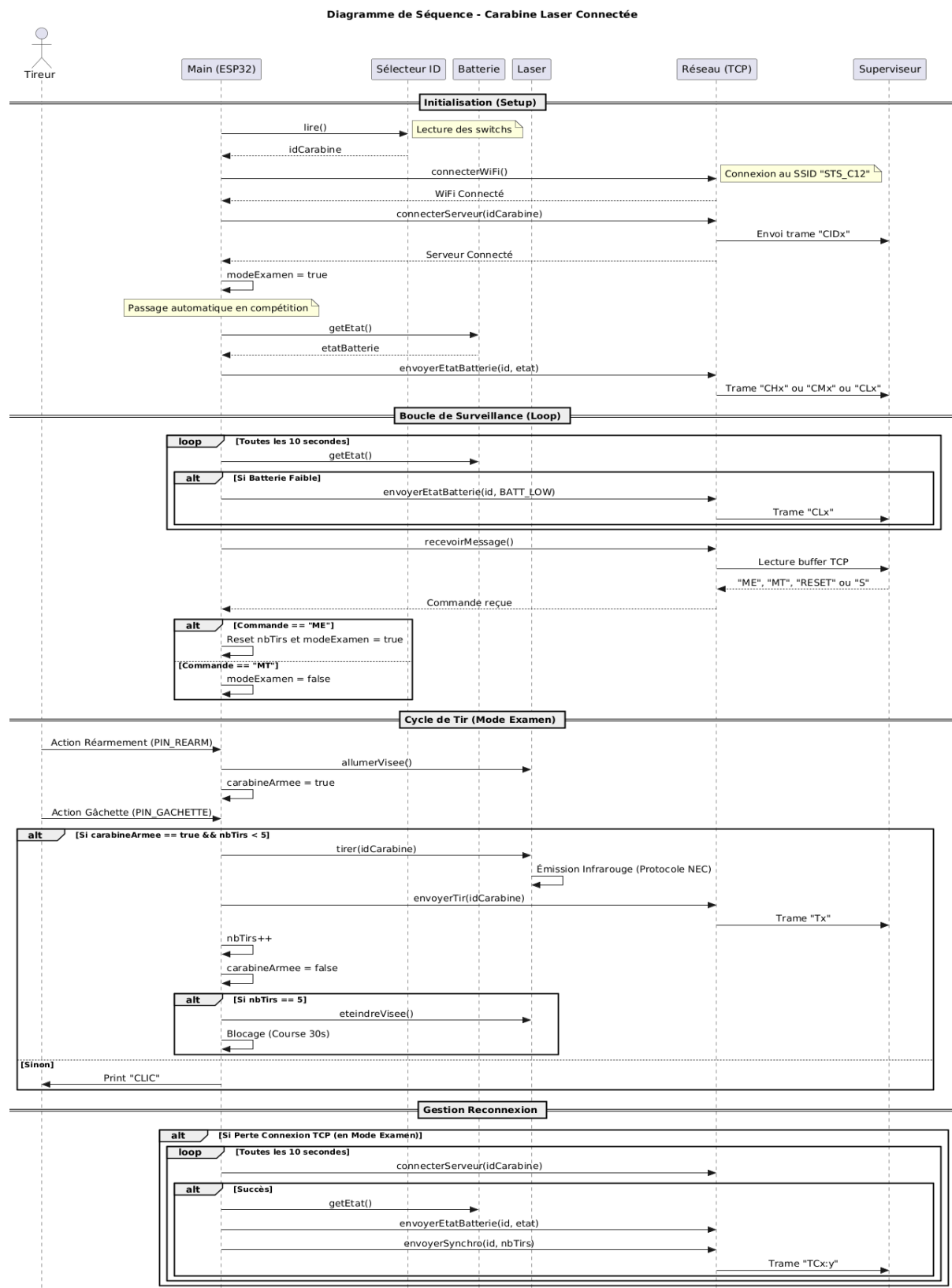
L'ESP32 sert de cerveau au dispositif. Il gère les entrées numériques (boutons) et les sorties (laser).



## 2.2 Diagramme de classes du code



## 2.3 Diagramme de séquence



## 2.4 Configuration des entrées/sorties (GPIO)

Toute la configuration des broches est centralisée dans le fichier 'Config.h'.

- **Laser** : Pin 4.
- **Gâchette** : Pin 6.
- **Réarmement** : Pin 5.
- **Sélecteur d'ID** : Pins 1, 2 et 3.
- **Batterie** : Pin 9.

```
1 #ifndef CONFIG_H
2 #define CONFIG_H
3
4 #define WIFI_SSID "STS_C12"
5 #define WIFI_PASSWORD "PervasioN"
6 #define SERVER_IP "192.168.4.54"
7 #define SERVER_PORT 2222
8
9 #define PIN_SW1 1
10 #define PIN_SW2 2
11 #define PIN_SW3 3
12 #define PIN_LASER 4
13 #define PIN_REARM 5
14 #define PIN_GACHETTE 6
15
16 #define PIN_BATTERIE 9
17 #endif
```

Dans ce fichier 'Config.h' on retrouve aussi les configurations du Wi-Fi et du serveur TCP.

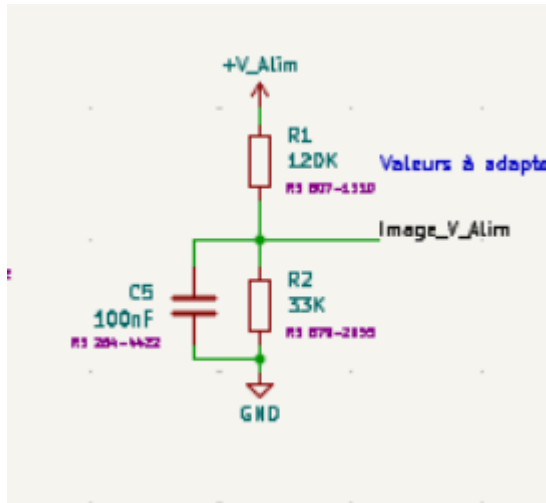
## 3. ÉTUDE PHYSIQUE : ACQUISITION DE LA TENSION BATTERIE

### 3.1 Problématique matérielle

Le microcontrôleur ESP32 fonctionne avec une tension de 3,3V. Cependant, la batterie utilisée pour alimenter la carabine délivre une tension beaucoup plus élevée (9V). Si cette tension était appliquée directement sur la broche analogique de l'ESP32, elle détruirait instantanément le composant. Il a donc fallu concevoir une interface physique d'adaptation avant de numériser le signal.

### 3.2 Le Pont Diviseur de Tension et de Protection

Pour abaisser la tension de la batterie à un niveau acceptable pour l'ESP32, la carte électronique intègre un circuit spécifique.



Ce circuit se compose de deux éléments clés :

1. **La diode de protection** : Elle sert à protéger le circuit contre les inversions de polarité au niveau du connecteur de batterie. Une diode induit naturellement une chute de tension physique d'environ **0,5V**.
2. **Le pont diviseur de tension (R1 et R2)** : Composé de deux résistances montées en série, il permet d'abaisser la tension restante.

La tension de sortie lue par la broche analogique de l'ESP32 est calculée par la formule du pont diviseur de tension :

$$V_{pin} = (V_{batterie} - V_{diode}) \times \frac{R_2}{R_1 + R_2}$$

Vpin étant la sortie du pont diviseur de tension.

Dans notre conception matérielle, avec  $R_1 = 120k\Omega$  et  $R_2 = 33k\Omega$ , nous obtenons un coefficient de réduction d'environ 4,64.

### 3.3 Le Convertisseur Analogique-Numérique (CAN)

La tension issue du pont diviseur est lue par le CAN de l'ESP32.

L'ESP32 possède un CAN d'une résolution de **12 bits**. Il convertit le signal analogique en une valeur numérique allant de 0 à 4095.

Le pas du CAN, pour une tension de référence de 3,3V, se calcule ainsi :

$$q = \frac{3.3}{4095} = 0.8 \text{ mV}$$

Cette précision est suffisante pour surveiller la décharge de la batterie.

### 3.4 Synthèse et Traitement Logiciel

Pour retrouver la véritable tension de la batterie dans le logiciel, le code effectue le cheminement physique inverse de notre schéma structurel.

1. **Conversion de la valeur binaire (0-4095) en Tension lue (Volts) :**

$$V_{pin} = \frac{\text{Valeur Brute} \times 3.3}{4095}$$

2. **Reconstruction mathématique de la tension initiale :**

On multiplie la tension de la broche par notre coefficient de pont diviseur (4,64), puis on réintègre la chute de tension perdue dans la diode (0,5V).

$$V_{batterie} = (V_{pin} \times 4.64) + 0.5$$

Cette modélisation mathématique est implémentée dans la méthode `lireTension()` de la classe `Batterie`, fournissant ainsi une valeur pour déterminer l'état de la batterie et envoyer les trames réseau (CL, CM, CH) au superviseur.

## 4. GESTION DE LA COMMUNICATION RÉSEAU

### 4.1 Connexion TCP et Wi-Fi

La classe 'Reseau' gère le réseau comme l'indique son nom. En phase de test, au démarrage, elle tente de se connecter au point d'accès Wi-Fi "STS\_C12" sinon en exercice réel, elle se connectera au point d'accès de la raspberry Pi. Une fois le Wi-Fi établi, elle ouvre une socket TCP vers le serveur superviseur sur le port 2222.

```
10 bool Reseau::connecterWiFi() {
11 Serial.println("\n--- Connexion WiFi ---");
12 WiFi.mode(WIFI_STA);
13 WiFi.begin(ssid, password);
14
15 int t = 0;
16 while (WiFi.status() != WL_CONNECTED && t < 20) {
17 delay(500);
18 Serial.print(".");
19 t++;
20 }
21
22 if (WiFi.status() == WL_CONNECTED) {
23 Serial.print("\nWiFi connecté ! IP : ");
24 Serial.println(WiFi.localIP());
25 return true;
26 }
27 return false;
28 }
29
30 bool Reseau::connecterServeur(int idCarabine) {
31 if (!client.connected()) {
32 Serial.printf("[TCP] Connexion au serveur %s:%d...\n", ip, port);
33 client.connect(ip, port);
34 }
35
36 if (client.connected()) {
37 client.printf("CID%d\n", idCarabine);
38 Serial.printf("[TCP] Connecté. Trame CID%d envoyée.\n", idCarabine);
39 return true;
40 }
41 return false;
42 }
```

## 4.2 Protocole de messagerie (Trames)

Dans le cadre des échanges entre une carabine et le superviseur, j'ai mis au point un protocole de communication basée sur des trames ASCII courtes afin de faire au plus simple.

### 1. Trames : Carabine vers Superviseur

Ces messages sont générés de manière autonome par la carabine et envoyés au PC. (Note : x représente l'ID physique de la cible configuré via le DipSwitch, par exemple 1, 2, 3...)

| Trame   | Événement déclencheur             | Description et Action côté Superviseur                                                                                                                                            |
|---------|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CIDx\n  | Connexion initiale ou Reconnexion | <b>Identification</b> : La carabine vient de se connecter au réseau TCP. Le superviseur sait maintenant que la carabine x est en ligne et prête.                                  |
| TCx:d\n | Envoyé lors d'une synchronisation | <b>Tir envoyé</b> : Envoyé lorsqu'on reçoit le signal de synchronisation, la carabine x envoie son nombre de tirs d.                                                              |
| Tx\n    | Tir envoyé                        | <b>Tir envoyé</b> : Envoyé lorsque la carabine x a effectué un tir.                                                                                                               |
| CLx\n   | Tension batterie < 7.4V           | <b>Battery Low (Batterie faible)</b> : Envoyé automatiquement toutes les 30 secondes si la carabine est sur le point de s'éteindre. Le superviseur doit <u>alerter</u> l'arbitre. |
| CMx\n   | 7.4V <= Tension batterie < 9V     | <b>Battery Mid (Batterie moyenne)</b> : Envoyé automatiquement au démarrage de la carabine si la batterie est à la moitié de sa capacité.                                         |

|       |                        |                                                                                                                       |
|-------|------------------------|-----------------------------------------------------------------------------------------------------------------------|
| CHx\n | Tension batterie >= 9V | <b>Battery High (Batterie Pleine)</b> : Envoyé automatiquement au démarrage de la carabine si la batterie est pleine. |
|-------|------------------------|-----------------------------------------------------------------------------------------------------------------------|

## 2. Trames : Superviseur vers Carabine

Ces commandes sont envoyées par le PC pour piloter les carabines à distance. La carabine lit les trames et cherche si le mot-clé est contenu dedans.

x = nombre de 0 ou 1

| Trame   | Nom de la commande           | Action exécutée par la Carabine                                                                                                                        |
|---------|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| ME\n    | Mode Examen                  | <b>Passage en Examen</b> : La carabine désactive son mode entraînement, et tirs limités à 5 avant une pause de 30 secondes.                            |
| MT\n    | Mode Training (Entraînement) | <b>Passage en Entraînement</b> : La carabine passe en mode entraînement, tirs illimités.                                                               |
| S\n     | Synchronisation              | <b>Synchronisation de l'état des tirs</b> : en cas de redémarrage de la carabine, le superviseur envoie une trame de synchronisation vers la carabine. |
| RESET\n | Réinitialisation             | <b>Remise à zéro</b> : Vide la mémoire des tirs.                                                                                                       |

## 5. Cybersécurité : Chiffrement et déchiffrement des trames TCP

Pour répondre aux exigences de sécurité du projet et empêcher l'interception ou la falsification des trames sur le réseau Wi-Fi local, j'ai développé et intégré un algorithme de chiffrement symétrique basé sur la fonction logique XOR (OU Exclusif) dans la classe `Reseau`.

**L'algorithme de chiffrement (`chiffreXOR`)** : Avant d'envoyer une trame sur le réseau TCP, la méthode `chiffreXOR` masque le message en clair à l'aide d'une clé secrète partagée (la chaîne de caractères `"TIR"`). Le programme boucle sur le message et applique l'opérateur bit à bit XOR (^) entre le caractère du message et celui de la clé. Pour que la clé s'adapte à n'importe quelle longueur de trame, j'utilise un index glissant avec l'opérateur modulo (`i % cle.length()`), ce qui permet de répéter la clé en boucle.

```
String Reseau::chiffreXOR(String message) {
 String cle = "TIR";
 String resultat = "";

 for (int i = 0; i < message.length(); i++) {
 // Application du OU Exclusif (XOR)
 char caractereChiffre = message[i] ^ cle[i % cle.length()];

 // Conversion en Hexadécimal
 char hex[3];
 sprintf(hex, "%02x", caractereChiffre);
 resultat += String(hex);
 }
 return resultat;
}
```

**L'encodage Hexadécimal (Protection du flux TCP)** : Appliquer un XOR sur des caractères ASCII génère régulièrement des caractères de contrôle non-imprimables (comme des sauts de ligne ou des caractères nuls). Ces valeurs risquent de corrompre l'objet `String` en C++ ou de tronquer la transmission TCP. Pour sécuriser le flux, le résultat du XOR est formaté systématiquement en une chaîne hexadécimale de deux caractères via la fonction `sprintf(hex, "%02x", caractereChiffre)`. Le texte envoyé sur le réseau est donc parfaitement stable.

**L'algorithme de déchiffrement (dechiffrerXOR) :** Le chiffrement XOR possède une propriété d'involution : appliquer deux fois le même masque redonne le message d'origine. À la réception d'un message du serveur, la méthode `dechiffrerXOR` parcourt la chaîne hexadécimale par blocs de deux caractères (`substring(i, i + 2)`). Elle convertit d'abord cette chaîne en véritable octet avec la fonction `strtol(blocHex.c_str(), NULL, 16)`, puis réapplique l'opérateur XOR avec la clé "TIR" (en recalculant l'index avec  $i / 2$ ). La carabine retrouve ainsi la trame en clair pour l'exécuter.

```
String Reseau::dechiffrerXOR(String messageHex) {
 String cle = "TIR";
 String resultat = "";
 for (int i = 0; i < messageHex.length(); i += 2) {
 String blocHex = messageHex.substring(i, i + 2);
 char octet = (char) strtol(blocHex.c_str(), NULL, 16);
 int indexLettre = i / 2;
 char caractereClair = octet ^ cle[indexLettre % cle.length()];
 resultat += caractereClair;
 }
 return resultat;
}
```

## 6. LOGIQUE MÉTIER ET MODES DE FONCTIONNEMENT

### 6.1 Gestion des actions physiques (Anti-rebond et États)

Pour la gâchette et le levier d'armement, je gère les événements matériels via des opérateurs booléens (`gachettePressee` et `rearmPresse`). Cela permet de détecter les fronts descendants (l'instant exact de l'appui par le tireur) et d'empêcher que la boucle `loop()` du microcontrôleur ne déclenche plusieurs tirs consécutifs tant que l'utilisateur maintient le doigt appuyé sur la gâchette.

```
18
19 // ETATS
20 bool carabineArmee = false;
21 bool gachettePressee = false;
22 bool rearmPresse = false;
```

### 6.2 Mode Entraînement

Dans ce mode, la liberté est totale. Le tireur peut tirer de manière illimitée. La seule contrainte est logicielle : après chaque tir, la variable `carabineArmee` passe à `false`, obligeant l'utilisateur à actionner physiquement le levier de réarmement pour tirer à nouveau

```
void gestionEntraînement() {
 bool actionGachette = (digitalRead(PIN_GACHETTE) == LOW);
 bool actionRearm = (digitalRead(PIN_REARM) == LOW);

 if (actionGachette && !gachettePressee) {
 if (carabineArmee) {
 nbTirs++;
 Serial.printf("[ENTRAINEMENT] Tir n°%d (Illimité)\n", nbTirs);
 laser.tirer(idCarabine);
 reseau.envoyerTir(idCarabine);
 carabineArmee = false;
 } else {
 Serial.println("CLIC (Armez la carabine)");
 }
 gachettePressee = true;
 delay(50);
 }

 if (actionRearm && !rearmPresse) {
 carabineArmee = true;
 laser.allumerVisee();
 Serial.println("CLAC-CLAC (Armé + Visée ON)");
 rearmPresse = true;
 delay(50);
 }

 if (!actionGachette) gachettePressee = false;
 if (!actionRearm) rearmPresse = false;
}
```

Le mode Examen impose des règles strictes :

1. **Limitation** : Le nombre de tirs est limité à 5.
2. **Verrouillage** : Une fois les 5 tirs effectués, la carabine se bloque.
3. **Course** : Une boucle de temporisation de 30 secondes bloque la carabine après les 5 tirs, le tireur doit aller courir. Durant cette phase, la carabine continue d'écouter le réseau pour permettre au superviseur de lancer une commande **RESET** ou **MT** .

```
void gestionExamen() {
 if (nbTirs >= 5) {
 laser.eteindreVisee();
 Serial.println("\n>>> FIN CHARGEUR <<<");
 for (int i = 30; i > 0; i--) {
 if(i % 5 == 0) {
 Serial.printf("Reste %d s...\n", i);
 }
 delay(1000);
 String msg = reseau.recevoirMessage();
 msg.trim();
 if (msg == "MT") {
 modeExamen = false;
 Serial.println(">>> REÇU 'MT' : FIN DE PAUSE, PASSAGE EN ENTRAINEMENT <<<");
 return;
 } else if (msg == "RESET") {
 nbTirs = 0;
 Serial.println(">>> REÇU 'RESET' : FIN DE PAUSE IMMEDIATE <<<");
 return;
 }
 }
 }
}
```

```

bool actionGachette = (digitalRead(PIN_GACHETTE) == LOW);
bool actionRearm = (digitalRead(PIN_REARM) == LOW);
if (actionGachette && !gachettePressee) {
 if (carabineArmee) {
 Serial.printf("[EXAMEN] Tir %d/5\n", nbTirs + 1);
 laser.tirer(idCarabine);
 reseau.envoyerTir(idCarabine);
 carabineArmee = false;
 nbTirs++;
 } else {
 Serial.println("CLIC (Armez la carabine)");
 }
 gachettePressee = true;
 delay(50);
}

if (actionRearm && !rearmPresse) {
 carabineArmee = true;
 laser.allumerVisee();
 rearmPresse = true;
 delay(50);
}

if (!actionGachette) gachettePressee = false;
if (!actionRearm) rearmPresse = false;
}

```

## 7. Robustesse du réseau : Le système de synchronisation

L'épreuve de biathlon se déroulant en extérieur, le risque de perte de la connexion Wi-Fi est un facteur critique qu'il a fallu anticiper. Pour rendre le système fiable et éviter de perdre des tirs en cas de déconnexion, j'ai mis en place un mécanisme de "synchronisation" des états.

- **Reconnexion non-bloquante** : Si la connexion au serveur est perdue, l'ESP32 ne se bloque pas. J'ai utilisé la fonction `millis()` pour tenter une reconnexion régulière (toutes les secondes) en arrière-plan, ce qui permet au tireur de continuer son épreuve sans latence sur la gâchette.
- **Synchronisation active** : Dès que le réseau est rétabli, la carabine force la mise à jour du serveur en lui poussant son véritable état (par exemple, la trame `TC3:4` pour informer que la carabine 3 a effectué 4 tirs au total).
- **Scrutation pendant la course** : En mode "Compétition", lorsque l'élève vide son chargeur (5 tirs), l'arme se verrouille pendant 60 secondes. Au lieu d'utiliser une instruction `delay(60000)` qui paralyserait totalement l'ESP32, j'ai conçu une boucle `for` de 60 itérations d'une seconde. À chaque itération, l'arme vérifie l'état du réseau, tente de se reconnecter si nécessaire, et écoute les messages du serveur (`recevoirMessage()`).
- **Ordres distants** : La carabine écoute en permanence le serveur. Si elle reçoit la commande spécifique `S` (Synchro), le programme intercepte l'ordre et exécute immédiatement la fonction `envoyerSynchro()`, renvoyant ainsi toutes ses variables à jour, même si elle est bloquée dans sa boucle de pénalité

```
void Carabine::gestionExamen() {
 if (nbTirs >= 5) {
 laser.eteindreVisée();
 Serial.println("\n>>> FIN CHARGEUR <<<");
 for (int i = 60; i > 0; i--) {
 if (i % 5 == 0) {
 Serial.printf("Reste %d s...\n", i);
 }
 if (WiFi.status() != WL_CONNECTED || !reseau.estConnecte()) {
 if (millis() - dernierEssaiReco >= 1000) {
 dernierEssaiReco = millis();
 Serial.println("⚠ Reconnexion pendant la pénalité...");
 if (WiFi.status() != WL_CONNECTED) {
 WiFi.reconnect();
 }
 } else if (!reseau.estConnecte()) {
 if (reseau.connecterServeur(idCarabine)) {
 Serial.println(">>> RECONNEXION TCP RÉUSSIE PENDANT LA PAUSE <<<");
 reseau.envoyerEtatBatterie(idCarabine, batterie.getEtat());
 reseau.envoyerSynchro(idCarabine, nbTirs);
 }
 }
 }
 }
 }
}
```

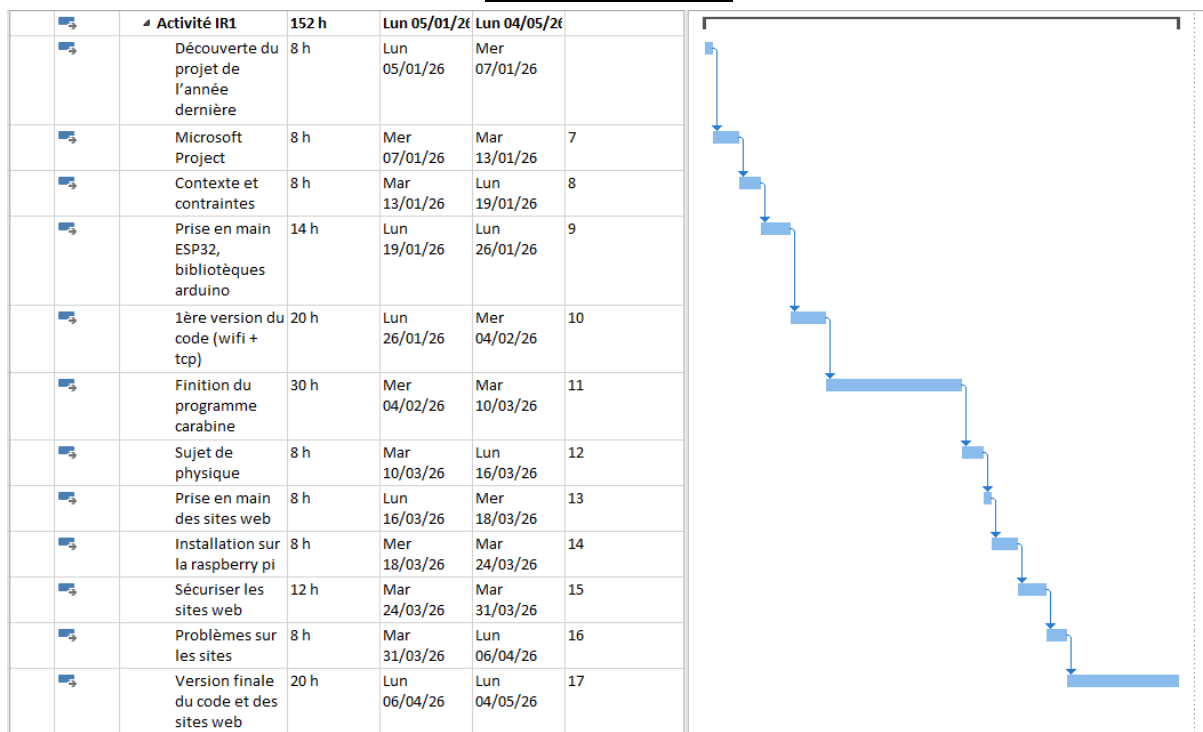
```

delay(1000);
String msg = reseau.recevoirMessage();
msg.trim();
if (msg == "MT") {
 modeExamen = false;
 Serial.println(">>> REÇU 'MT' : FIN DE PAUSE, PASSAGE EN ENTRAÎNEMENT <<<");
 return;
} else if (msg == "RESET") {
 nbTirs = 0;
 Serial.println(">>> REÇU 'RESET' : FIN DE PAUSE IMMEDIATE <<<");
 return;
}
else if (msg == "S") {
 reseau.envoyerSynchro(idCarabine, nbTirs);
 Serial.printf(">>> ORDRE REÇU PENDANT PAUSE : SYNCHRO (Envoi TC%d:%d) <<<\n", idCarabine, nbTirs);
}
}
nbTirs = 0;
Serial.println(">>> FIN DE LA COURSE : CHARGEUR RÉINITIALISÉ <<<");
}

```

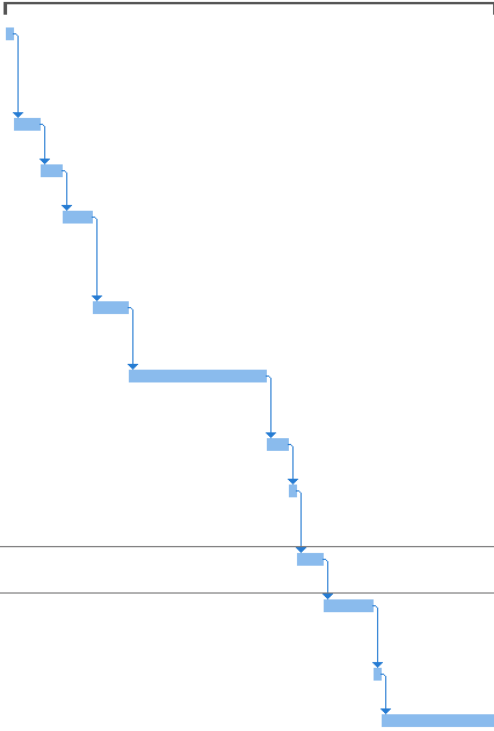
## 8.CALENDRIER

### Prévisionnel



## Réel

|  | Activité IR1                               | 160 h | Lun 05/01/26 | Mer 06/05/26 |    |
|--|--------------------------------------------|-------|--------------|--------------|----|
|  | Découverte du projet de l'année dernière   | 8 h   | Lun 05/01/26 | Mer 07/01/26 |    |
|  | Microsoft Project                          | 8 h   | Mer 07/01/26 | Mar 13/01/26 | 7  |
|  | Contexte et contraintes                    | 8 h   | Mar 13/01/26 | Lun 19/01/26 | 8  |
|  | Prise en main ESP32, bibliothèques arduino | 14 h  | Lun 19/01/26 | Lun 26/01/26 | 9  |
|  | 1ère version du code (wifi + tcp)          | 20 h  | Lun 26/01/26 | Mer 04/02/26 | 10 |
|  | Avancée du programme carabine              | 30 h  | Mer 04/02/26 | Mar 10/03/26 | 11 |
|  | Sujet de physique                          | 8 h   | Mar 10/03/26 | Lun 16/03/26 | 12 |
|  | 2ème version du code (communication)       | 8 h   | Lun 16/03/26 | Mer 18/03/26 | 13 |
|  | Organisation en classes                    | 8 h   | Mer 18/03/26 | Mar 24/03/26 | 14 |
|  | Synchronisation carabine cible raspberry   | 20 h  | Mar 24/03/26 | Lun 06/04/26 | 15 |
|  | Problèmes sur le code                      | 8 h   | Lun 06/04/26 | Mer 08/04/26 | 16 |
|  | Version finale du code                     | 20 h  | Mer 08/04/26 | Mer 06/05/26 | 17 |



## 9. CONCLUSION

Le système développé est fonctionnel. Il respecte les contraintes de sécurité du tir laser et assure une remontée d'information précise vers le superviseur. L'architecture en classes a permis de produire un code propre et évolutif. Au niveau du calendrier prévu je n'ai pas pu passer du temps sur les sites web car j'ai dû consacrer énormément de temps sur le code de la carabine.

## FICHE RECETTE, QUALIFICATION DU SYSTÈME

|                  |                      |                               |            |
|------------------|----------------------|-------------------------------|------------|
| Nom du système : | BIATHLON             | Code de la campagne de test : |            |
| Technicien 1 :   | Spinosi-baumet Lucas | Date :                        | 30/05/2026 |
| Technicien 2 :   | Retamosa thomas      |                               |            |
| Technicien 3 :   | El Allouchi Youssef  |                               |            |
| Technicien 4 :   | Klipel Timéo         |                               |            |
| Technicien 5 :   | Parisis Alexandre    |                               |            |

### IDENTIFICATION DU SCENARIO

| Identification du scénario de recette |                                                                                                                                                                              |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Titre :                               | Détection, validation d'un tir réussi et synchronisation réseau/BDD                                                                                                          |
| Objectif du scénario :                | Vérifier que la réception d'une trame de tir ("T3") suivie d'un impact ("H") incrémente le score, met à jour la base de données et ordonne l'allumage des LEDs sur la cible. |

### CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

- Le serveur logiciel QbiathlonV6 est démarré et écoute sur le port 2222
- La base de données MySQL locale (biathlon) est accessible et l'état de la course (state) vaut 3 (Course lancée).
- La Carabine est connecté (trame d'identification TIDX envoyée).
- La Cible est connecté (trame d'identification CIDX envoyée).

### EXÉCUTION DU TEST

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description courte : | 1 - Déclenchement du tir : La carabine envoie la trame (chiffree en XOR).<br>2 - Détection de l'impact : Dans le délai de 200 ms, la cible envoie la trame "H" (Hit).<br>3 - Validation (Fin temporisation) : Le timer de 200 ms expire.<br>4 - Retour Visuel : Le serveur notifie l'équipement physique de la cible.<br>5 - Persistance BDD : Vérification de l'enregistrement du score dans MySQL.                                                                                                                                                                                                           |
| Résultats attendus : | 1 - Le superviseur décrypte "Tx", identifie la cible visée et lance une temporisation de 200 ms.<br>2 - Le superviseur reçoit "H" et passe immédiatement la variable cibleAtteinte à true dans la mémoire partagée (CZdc).<br>3 - Le serveur lit cibleAtteinte == true, appelle validerTir(true) et affiche dans la console : "TIR DÉTECTÉ - ÉQUIPEMENT - TOUCHÉ".<br>4 - Une trame contenant la nouvelle séquence (ex: "1") est envoyée en clair via le socket de la cible.<br>5 - Une requête UPDATE a bien été exécutée sur la table race. Le champ seqTirs1 contient désormais la valeur "1/1" avec l'ID x |

### BILAN

Le système a parfaitement validé le tir, le score en base est correct et la LED s'est allumée instantanément.

### REMARQUES

Penser à vérifier le comportement si le tir arrive alors que l'état de la course n'est pas à 3, le tir doit être ignoré comme spécifié dans le code de validerTir.

### CONCLUSION

|        |          |            |  |
|--------|----------|------------|--|
| VALIDE | <b>X</b> | NON VALIDE |  |
|--------|----------|------------|--|

(Mettre une croix dans la case correspondante)

# **Alexandre Parisis**

*Partie Personnelle- Alexandre Parisis*



|                                         |       |
|-----------------------------------------|-------|
| 1. INTRODUCTION ET OBJECTIFS DU PROJET  | 3-6   |
| 1.1 Cahier des charges personnel        | 4     |
| 1.2 Planification et Pilotage du Projet | 5     |
| 1.3 Problemes rencontrés                | 6     |
| <u>2. ANALYSE TECHNIQUE</u>             | 7-23  |
| <u>3. PROCESSUS DE FABRICATION</u>      | 23-29 |
| 4. ANNEXES                              | 30-31 |

## 1 - Introduction au projet

Le projet Biathlon consiste à créer un équipement moderne en réponse à la demande du professeur d'EPS Monsieur LeDuc, afin de proposer une activité de biathlon comme épreuve au baccalauréat.

L'objectif est de développer une solution simple à utiliser, peu coûteuse et proche du matériel professionnel. Le système est aussi pensé pour être facilement reproductible grâce aux fichiers de fabrication (fichiers Gerber).

Le projet comprend trois éléments principaux : une carabine laser, une cible électronique et un logiciel avec une interface de gestion pour contrôler l'ensemble.

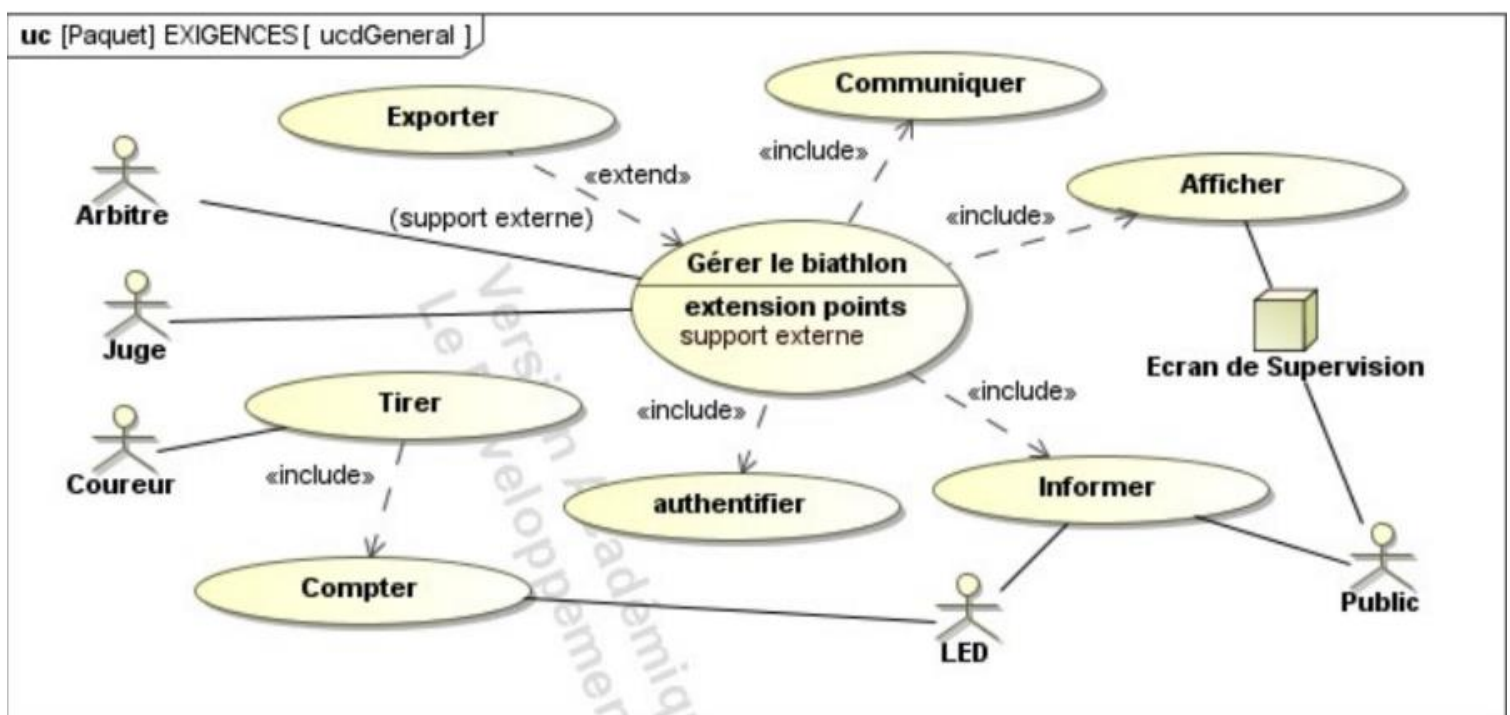


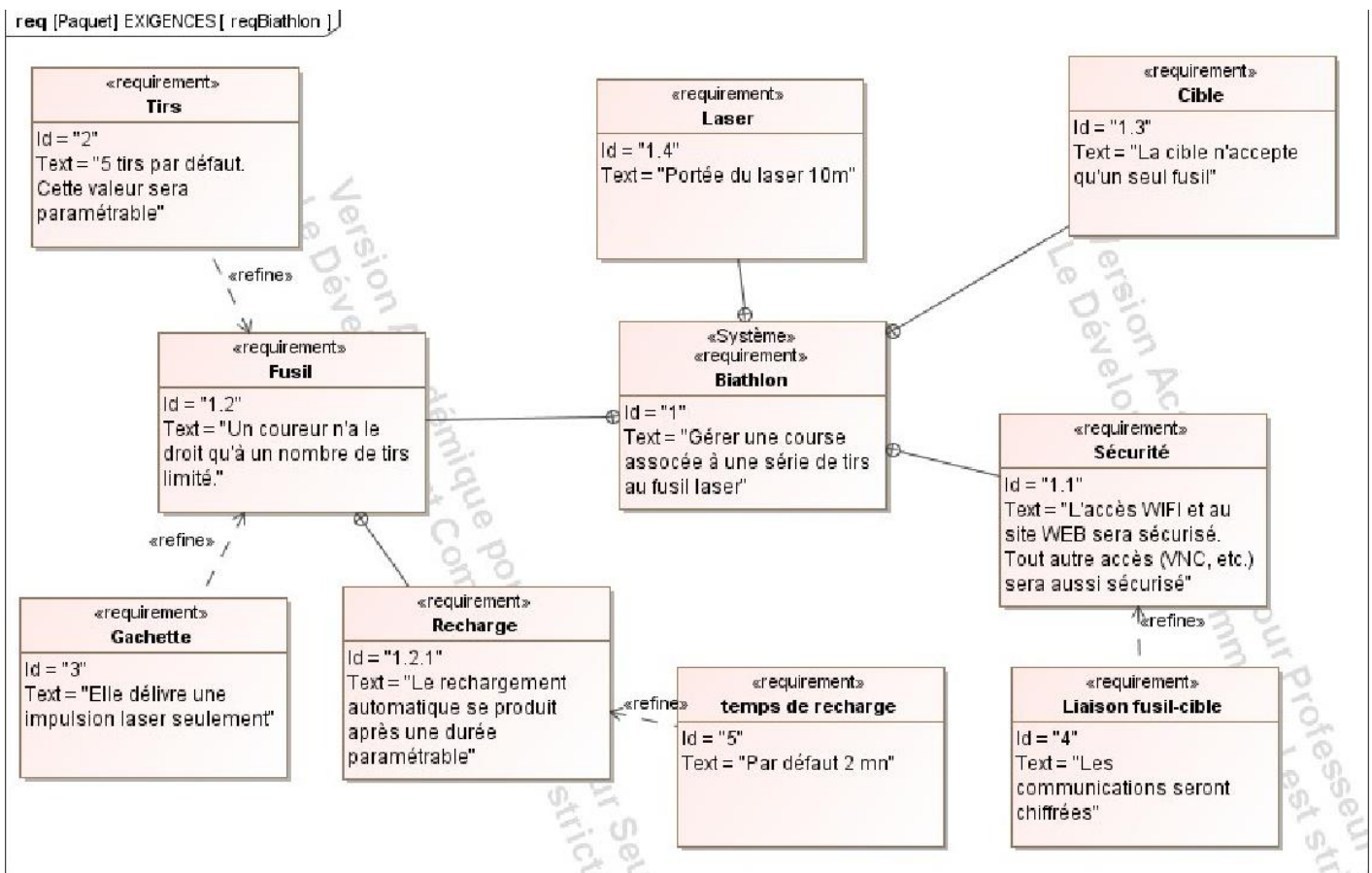
Diagramme SysML "Cas d'utilisation"

## 1.1 Cahier des charges personnel

L'objectif principal est de concevoir une nouvelle architecture matérielle pour la carte électronique de la carabine. Comme il s'agit d'un projet repris de l'année précédente, nous développons ici une version modifiée et adaptée.

Dans ce cadre, mes missions sont les suivantes :

- **Analyse de la version 2025** : Étudier le dossier technique et l'architecture de la carte existante afin de comprendre son fonctionnement et identifier les améliorations possibles.
- **Optimisation** : Faire évoluer le schéma électronique en simplifiant le circuit, en réduisant l'encombrement et en améliorant l'autonomie.
- **Fonctionnalités supplémentaires** : Ajouter un indicateur de niveau de batterie par LED (allumage momentané)



## 1.2 Planification et Pilotage du Projet

Diagramme de Gantt (prévisionnel) :

| Nom de la tâche ▼                                          | Durée ▼      | Début ▼             |
|------------------------------------------------------------|--------------|---------------------|
| ▲ <b>Projet</b>                                            | <b>104 h</b> | <b>Mar 06/01/26</b> |
| ▲ <b>Prise en main commune</b>                             | <b>16 h</b>  | <b>Mar 06/01/26</b> |
| Découverte du projet et de l'ancien dossier                | 4 h          | Mar 06/01/26        |
| Création du planning visé                                  | 8 h          | Lun 12/01/26        |
| Approfondisse de l'étude du                                | 4 h          | Lun 19/01/26        |
| ▲ <b>Partie ER 2</b>                                       | <b>92 h</b>  | <b>Lun 19/01/26</b> |
| Analyse et teste de la carte électronique carabine         | 20 h         | Lun 19/01/26        |
| Modifications structurelles et optimisation de cette carte | 20 h         | Mar 03/02/26        |
| Routage carte                                              | 20 h         | Mar 10/03/26        |
| cablage et essai sur la carte                              | 12 h         | Lun 30/03/26        |
| Documentation de la mise en service et maintenance         | 20 h         | Mar 28/04/26        |

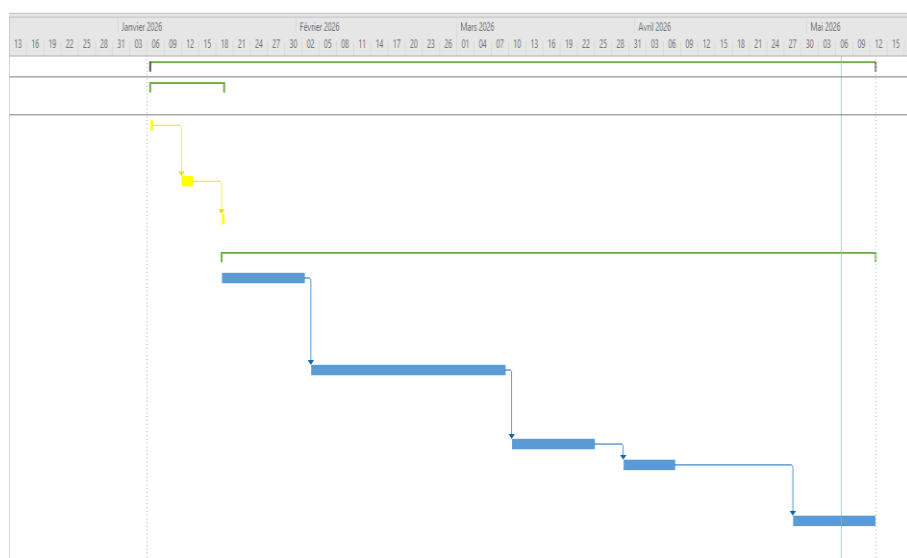


Diagramme de Gantt (final) :

| Nom de la tâche                                      | Durée        | Début               | Fin                 |
|------------------------------------------------------|--------------|---------------------|---------------------|
| ▼ <b>Projet</b>                                      | <b>140 h</b> | <b>Mar 06/01/26</b> | <b>Mar 16/06/26</b> |
| ▼ <b>Prise en main commune</b>                       | <b>12 h</b>  | <b>Mar 06/01/26</b> | <b>Mar 13/01/26</b> |
| — Découverte du projet et de l'ancien dossier        | 4 h          | Mar 06/01/26        | Mar 06/01/26        |
| — Création du planning prévisionnel                  | 4 h          | Lun 12/01/26        | Lun 12/01/26        |
| — Approfondissement de l'étude du projet             | 4 h          | Mar 13/01/26        | Mar 13/01/26        |
| ▼ <b>Partie 2 : modification et avancé</b>           | <b>128 h</b> | <b>Lun 19/01/26</b> | <b>Mar 16/06/26</b> |
| — Analyse et tests de la carte électronique carabine | 24 h         | Lun 19/01/26        | Mar 03/02/26        |
| — Modifications structurelles et optimisation        | 20 h         | Mar 03/02/26        | Lun 09/03/26        |
| — Routage de la carte                                | 36 h         | Mar 10/03/26        | Mar 21/04/26        |
| — Câblage, soudage et essais                         | 16 h         | Mar 28/04/26        | Mar 19/05/26        |
| └ Soudage des composants (CMS et traversants)        | 6 h          | Mar 28/04/26        | Mer 06/05/26        |
| └ Essais sur carte et validation des fonctions       | 10 h         | Jeu 07/05/26        | Mar 19/05/26        |
| — Documentation et livrables                         | 32 h         | Lun 25/05/26        | Mar 16/06/26        |
| └ Rédaction du rapport technique de projet           | 32 h         | Lun 25/05/26        | Mar 16/06/26        |

Dans l'ensemble, j'ai réussi à maintenir le rythme de travail imposé par mon diagramme de Gantt initial. La répartition des tâches entre les phases de recherche et d'analyse (janvier, février), de conception (février/mars) et de réalisation (avril/mai) a été globalement respectée, me permettant d'arriver à la revue actuelle avec une carte quasi finalisée.

### **1.3 Problèmes rencontrés :**

#### **Saisie du schéma structurel :**

Cette étape a été prolongée suite à la nécessité de mettre à jour les bibliothèques de composants (librairies KiCad) et le fait de partir d'un schéma déjà existant (schéma de carte utilisée pour plusieurs projets) et de devoir le modifier m'a pris plus de temps que prévu.

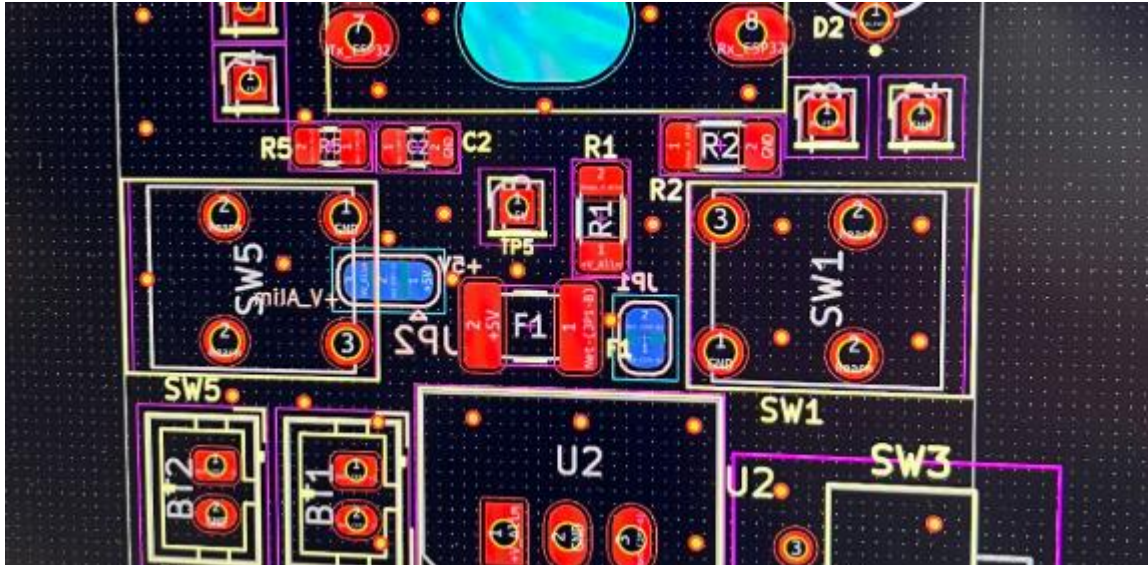
#### **Optimisation du routage PCB :**

Le routage a été plus chronophage que prévu. Des changements de dernière minute sur la structure de la carte électronique de la carabine m'ont obligé à revoir le placement des composants pour gagner de la place et pour être le plus optimal possible. Ce travail de "re-routage" était indispensable pour garantir que la carte s'insère parfaitement dans le boîtier.

#### **Nouvelle demande :**

Pendant cette même période, M. Leduc a souhaité modifier le système de réarmement, qui se trouvait uniquement sur un côté de la carte. Cela posait un problème d'équité entre les utilisateurs droitiers et gauchers. La solution retenue a été d'installer non pas un, mais deux boutons poussoirs, placés de chaque côté de la carte afin de faciliter son utilisation pour tous les utilisateurs. (schéma page)

*(voici une image agrandis du plan de la carte ou se trouve les 2 boutons poussoir)  
(SW5 et SW1 étant les 2 boutons poussoir disposés de parts et d'autre de la carte) :*

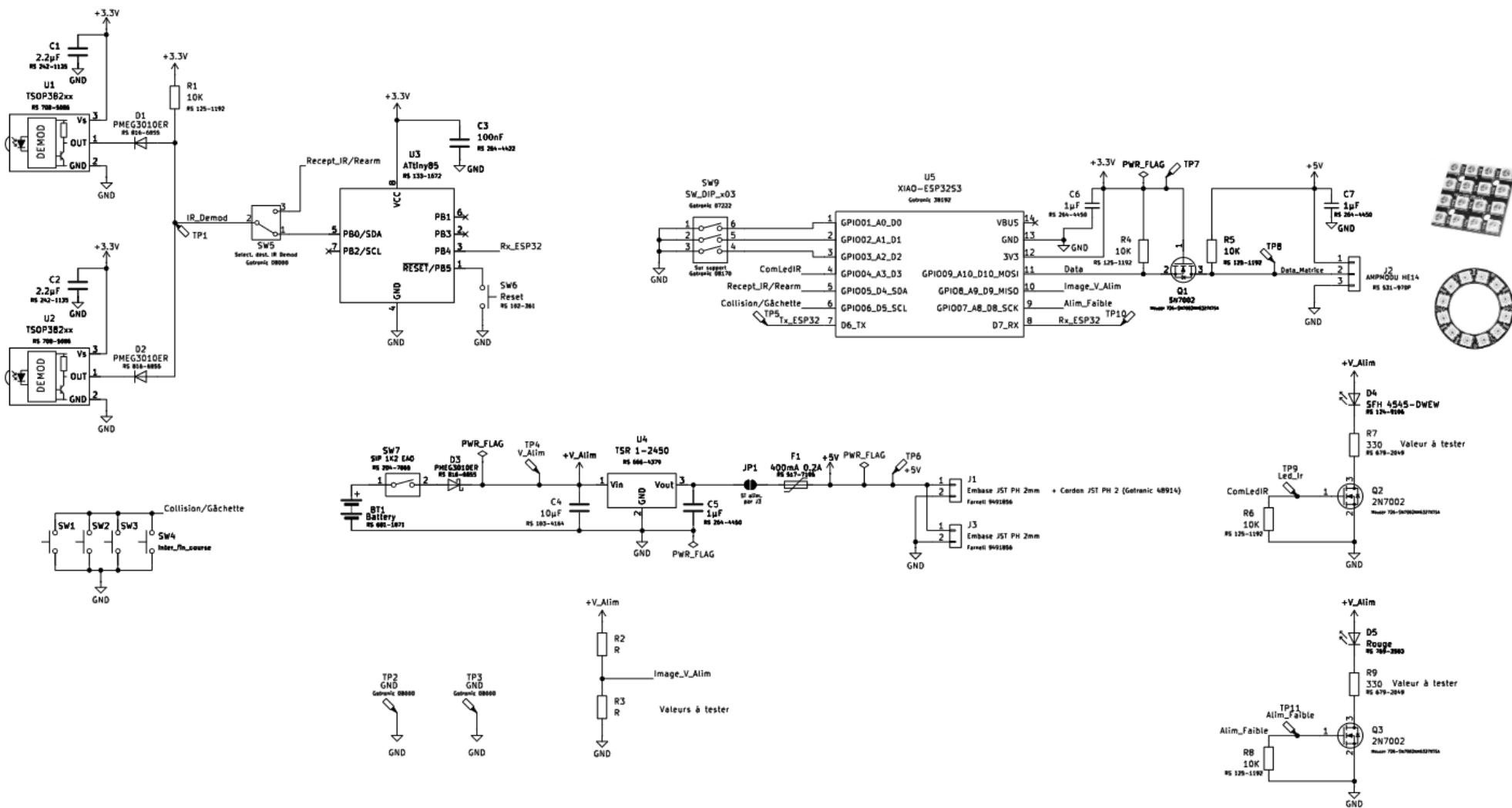


## PARTIE 2, ÉTUDE TECHNIQUE:

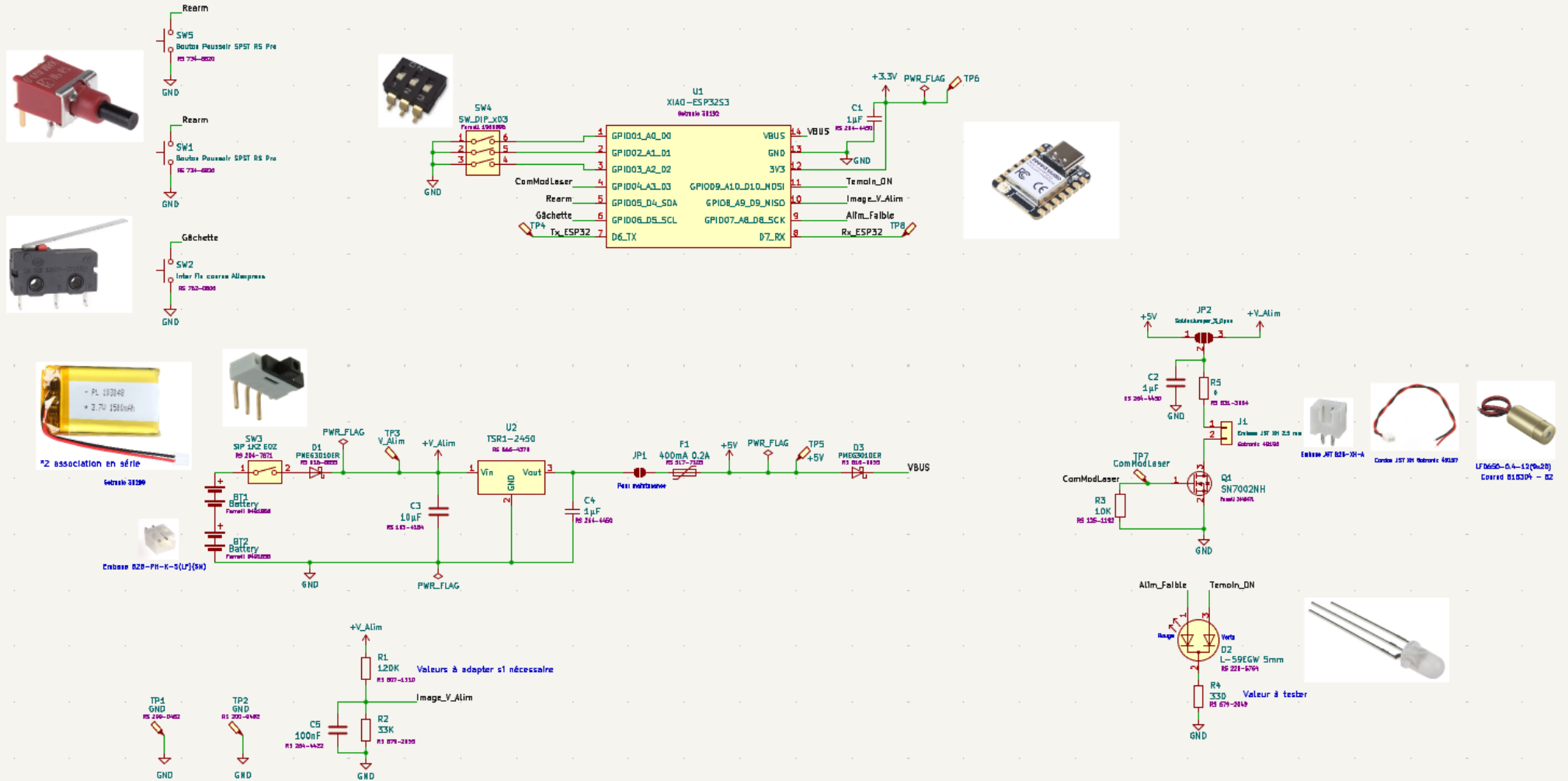
Au début du projet en janvier, j'ai analysé la carte électronique utilisée l'année dernière (mise à disposition pour 2 projets). Même si elle fonctionnait, elle ne correspondait plus à nos besoins pour plusieurs raisons :

- Elle était trop volumineuse : Avec la nouvelle carabine plus compacte, la carte ne rentrait plus dans le boîtier.
- manque d'une LED : En effet, suite à la première version de 2025, il a été convenu de rajouter une LED bicolore pour signaler l'état de la batterie à l'utilisateur.
- Il y avait trop de composants inutiles : La carte de l'an dernier pouvait aussi "recevoir" des signaux, alors que notre carabine doit seulement "envoyer" le laser par exemple.

## Schéma structurel 2025



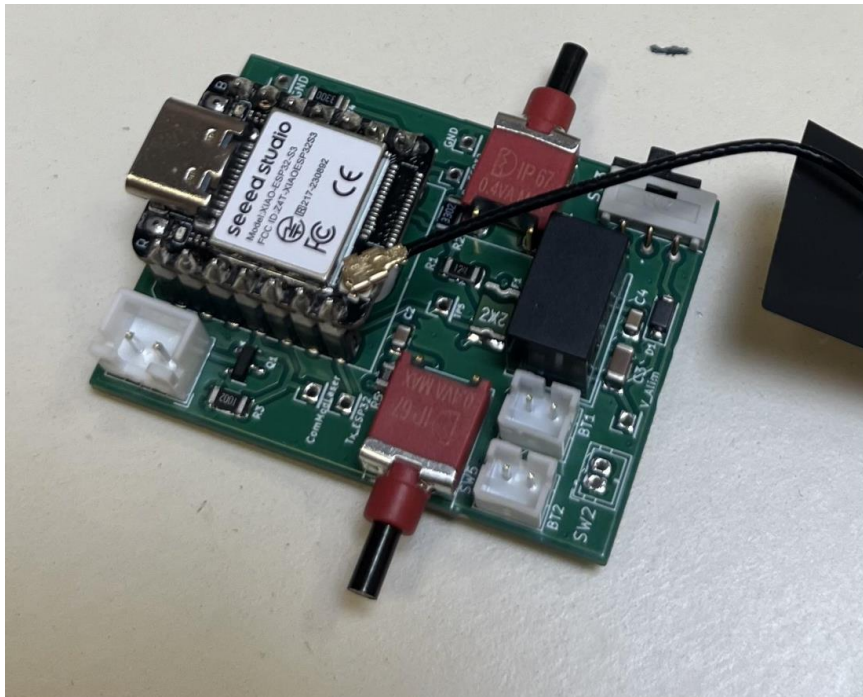
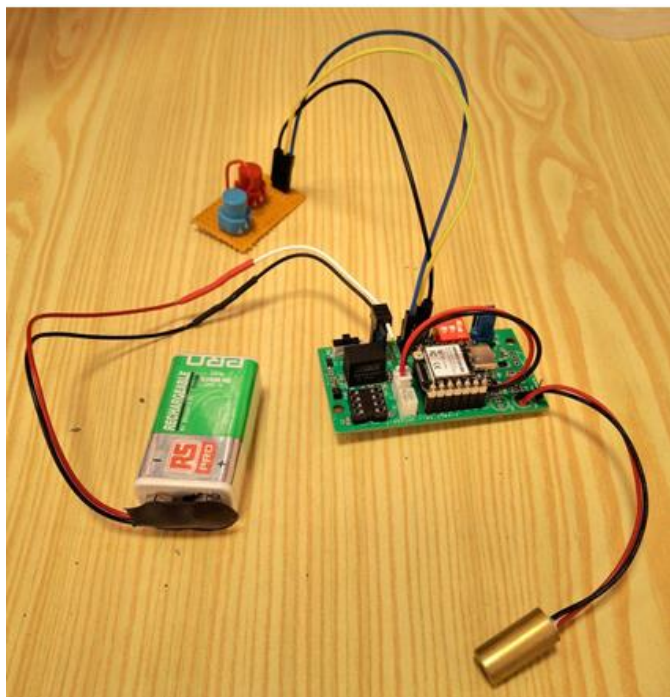
# Schéma structurel 2026



**La version 2026 est plus adaptée car :**

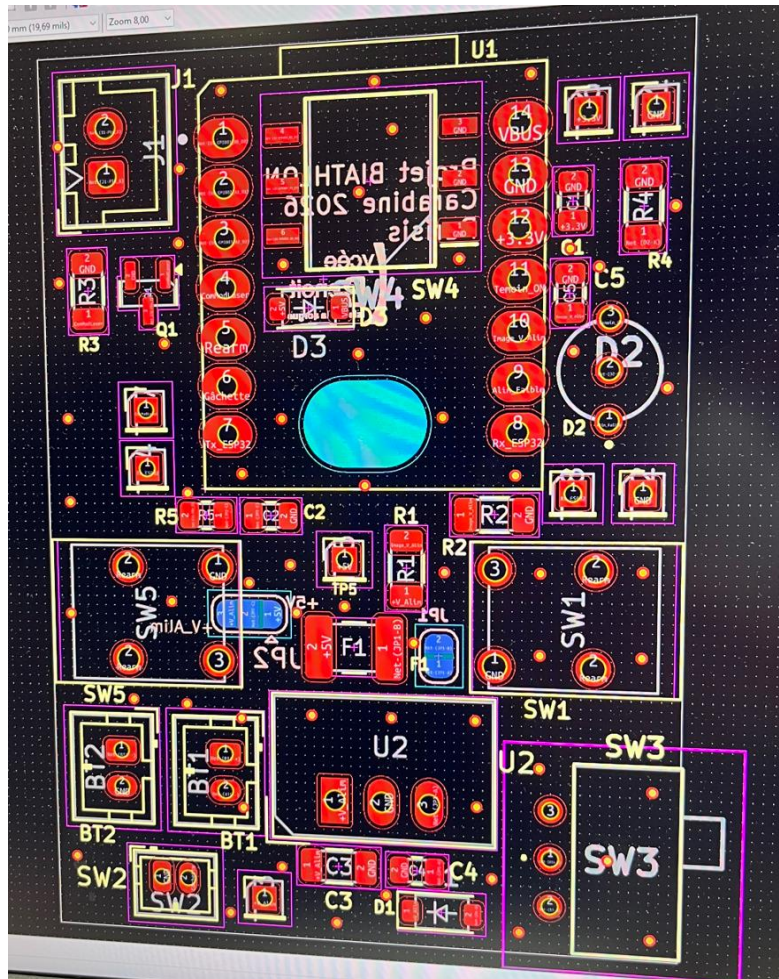
La surface de la carte a été drastiquement réduite afin de s'intégrer parfaitement dans le nouveau boîtier de la carabine. L'ancienne version était trop encombrante, ce qui posait des problèmes lors de l'assemblage mécanique. Pour réussir cette miniaturisation, j'ai entièrement réorganisé les éléments du schéma et optimiser le placement de chaque composant pour réduire l'encombrement final sur le PCB.

*Photo carte avant / après :*



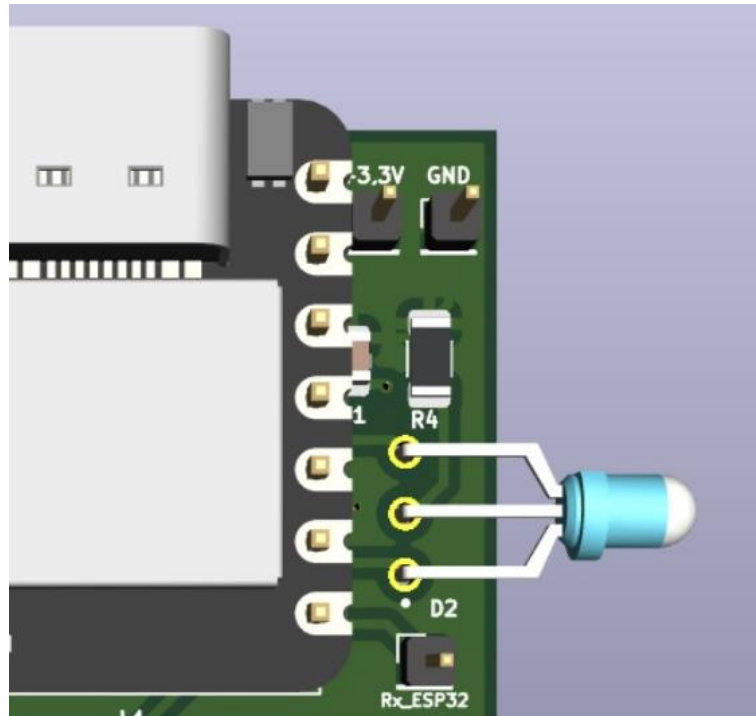
En complément de ce gain de place, j'ai effectué un travail d'optimisation sur le schéma structurel en supprimant tous les éléments inutiles à notre projet, comme les modules de réception. Cette simplification rend la carte plus fiable et moins gourmande en énergie. La conception est désormais concentrée sur l'émission laser pure, ce qui limite les risques de panne et rend le routage beaucoup plus propre.

## Nouvelle organisation de la carte après toutes les modifications demandées :



Cette nouvelle version apporte également des fonctionnalités qui manquaient sur le précédent modèle. J'ai ajouté une LED bicolore qui permet de connaître l'état de la batterie en temps réel. Ce signal visuel améliore le confort d'utilisation et permet d'anticiper la recharge avant une séance de tir.

*(image agrandis du placement de la led bicolore sur le modèle 3d)*



**Point à noter:** Mes camarades de la partie IR ont fait des tests avec l'ancienne carte de 2025 et la batterie NiMH prévue au départ (Tension : 9 V / Capacité : 420 mAh). Le constat a été rapide : la batterie se vidait beaucoup trop vite.

Pour régler ce problème, nous utilisons maintenant deux batteries LiPo branchées en série (Tension : 3.7 Vcc / Capacité : 1500 mAh).

**Le tableau comparatif ci-dessous met en évidence les caractéristiques des deux solutions :**

| <b>Caractéristiques</b>    | <b>Ancien modèle</b>                     | <b>Nouveau modèle</b>              |
|----------------------------|------------------------------------------|------------------------------------|
| <b>Technologie</b>         | NiMH (Nickel-Métal Hydrure) - Pile 6LR61 | LiPo (Lithium-Polymère) - THYM-BAT |
| <b>Tension</b>             | 8,4 V (nominale) / 9 V                   | 3,7 Vcc                            |
| <b>Capacité</b>            | 250 mAh                                  | <b>1500 mAh</b>                    |
| <b>Prix unitaire (TTC)</b> | <b>5,49 €</b>                            | <b>19,20 €</b>                     |

À première vue, passer au nouveau modèle semble représenter un investissement initial beaucoup plus élevé. La batterie LiPo coûte 19,20 €, soit environ 3,5 fois le prix de l'ancienne batterie NiMH qui était de 5,49 €.

Cependant, ce surcoût est largement compensé par des avantages techniques importants pour le bon fonctionnement et la viabilité du projet.

Le nouveau modèle offre une autonomie six fois supérieure à l'ancien. Cela signifie que la nouvelle batterie dure beaucoup plus longtemps et nécessite moins de recharges.

De plus, la batterie LiPo est plus petite et plus légère que l'ancienne, ce qui permet une intégration parfaite dans l'ensemble mécanique sans sacrifier la puissance.

Enfin, les batteries LiPo ont une durée de vie plus longue et sont plus rentables à long terme. Elles supportent plus de cycles de charge et de décharge que les anciennes batteries NiMH. Cela signifie que les composants devront être remplacés moins souvent, ce qui amortit l'écart de prix initial sur la durée de vie du produit.

***(image des nouvelles batteries utilisées dans le boîtier) :***



***(image de l'ancienne batterie utilisée dans le boîtier) :***



## Validation de la portée et test du laser :

Le choix du laser de Classe 1 (0,4 mW) a été fait pour garantir une sécurité totale pour les yeux des utilisateurs, tout en conservant une puissance suffisante pour l'exercice. Pour valider ce choix, j'ai réalisé un test de portée en conditions réelles afin de vérifier si le faisceau restait précis et détectable à la distance réglementaire du biathlon, soit 10 mètres.

### *(module laser)*

#### Classification de pointeur laser - laserpuissant.com



##### Classification

Les installations à laser (fixes et mobiles) sont répertoriées selon le potentiel de risques que présentent leurs rayonnements dans des classes fixées selon la norme EN 60825-1. Ces classes, au nombre de quatre, sont encore divisées en sous-classes (cf. tableau 1).

Seuls les laser jusqu'à la classe 2 sont inoffensifs (cf. également Dangers et risques pour la santé). ).

Il est donc recommandé de n'utiliser que des **pointeurs laser** des classes 1 et 2, dont la puissance maximale ne dépasse pas 1 mW. Les lasers de puissance supérieure sont dangereux et nécessitent des installations et des mesures de sécurité supplémentaires. Les **lasers des classes 3B et 4** ne peuvent être utilisés que de manière stationnaire.

| Etiquetage européen | Etiquetage américain | Puissance typique en milliwatt (mW) | Exemples d'applications          |
|---------------------|----------------------|-------------------------------------|----------------------------------|
| Classe 1            | Classe I             | < 0,4 mW                            | scanners de caisse, lecteurs DVD |
| Classe 2            | Classe II            | < 1 mW                              | Pointeurs laser                  |
| Classe 3R           | Classe IIIa          | < 5 mW                              | Lasers de spectacles             |
| Classe 3B           | Classe IIIb          | < 500 mW                            | Lasers de spectacles             |
| Classe 4            | Classe IV            | > 500 mW                            | Lasers de spectacles             |

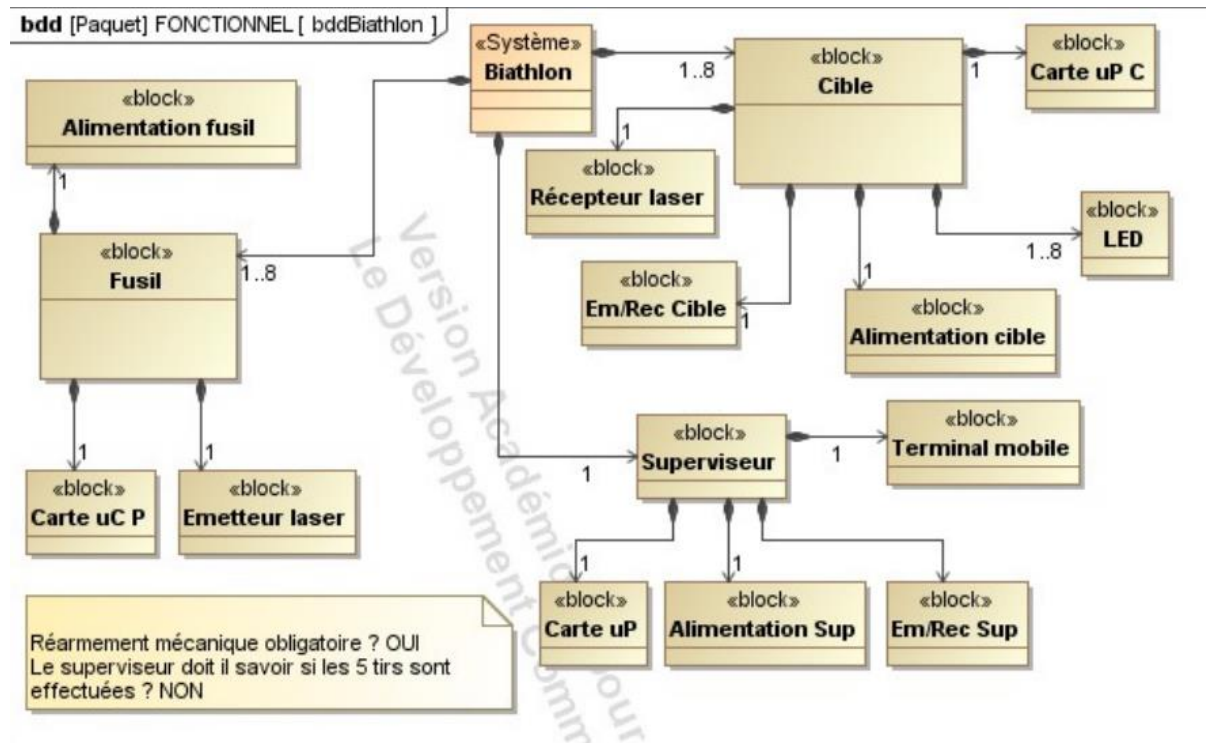
Lasers et systèmes de laser sont affectés l'un des quatre grandes classes (I à IV) en fonction du potentiel de causer des dommages biologiques. Le fondement biologique des classes de danger sont résumés dans le tableau ci-dessous:

| Classe 1                            | Classe 2                                                                                                                          | Classe 3A                                                                                                                            |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| Lasers intrinsèquement sans danger. | Lasers de faible puissance (<1 mW) émettant dans le visible (400 à 700 nm). Protection de l'œil assurée par le réflexe palpébral. | Lasers de puissance moyenne (<5 mW). Vision directe dangereuse si elle est supérieure à 0,25 s ou à travers un instrument d'optique. |
| Appareil à laser de classe 1.       | Rayonnement laser<br>Ne pas regarder le faisceau<br>Appareil à laser de classe 2.                                                 | Rayonnement laser<br>Ne pas regarder le faisceau à l'œil nu ou avec un instrument d'optique<br>Appareil à laser de classe 3A.        |



L'intégration du module laser ajoute deux contraintes majeures au projet :

- Un prix élevé : Affiché à 22,49 € TTC l'unité, ce composant représente un coût supérieur à celui de la nouvelle batterie.
- Un stock limité : Avec seulement 91 pièces disponibles chez le fournisseur (Conrad, mai 2026), le risque de rupture de stock est réel et pourrait bloquer la production. Un achat anticipé est donc indispensable.

**L'unité de traitement:**

**Le choix du module XIAO** est justifié par son rapport performance/taille. Contrairement à une carte ESP32 classique ou une carte Arduino, le module est très petit. Cette caractéristique était indispensable pour respecter les contraintes du de taille, où l'espace disponible est très limité.

Techniquement, ce module est préférable car il intègre nativement le Wi-Fi, ce qui évite l'ajout de composants de communication externes encombrants. Il dispose de suffisamment d'entrées/sorties pour gérer simultanément le laser, la gâchette et la surveillance de la batterie par LED.

Le connecteur USB-C intégré permet également de programmer le module et de le manipuler plus facilement.

*(image ESP32-S3)*



*(image ESP32-S3 sur la carte)*



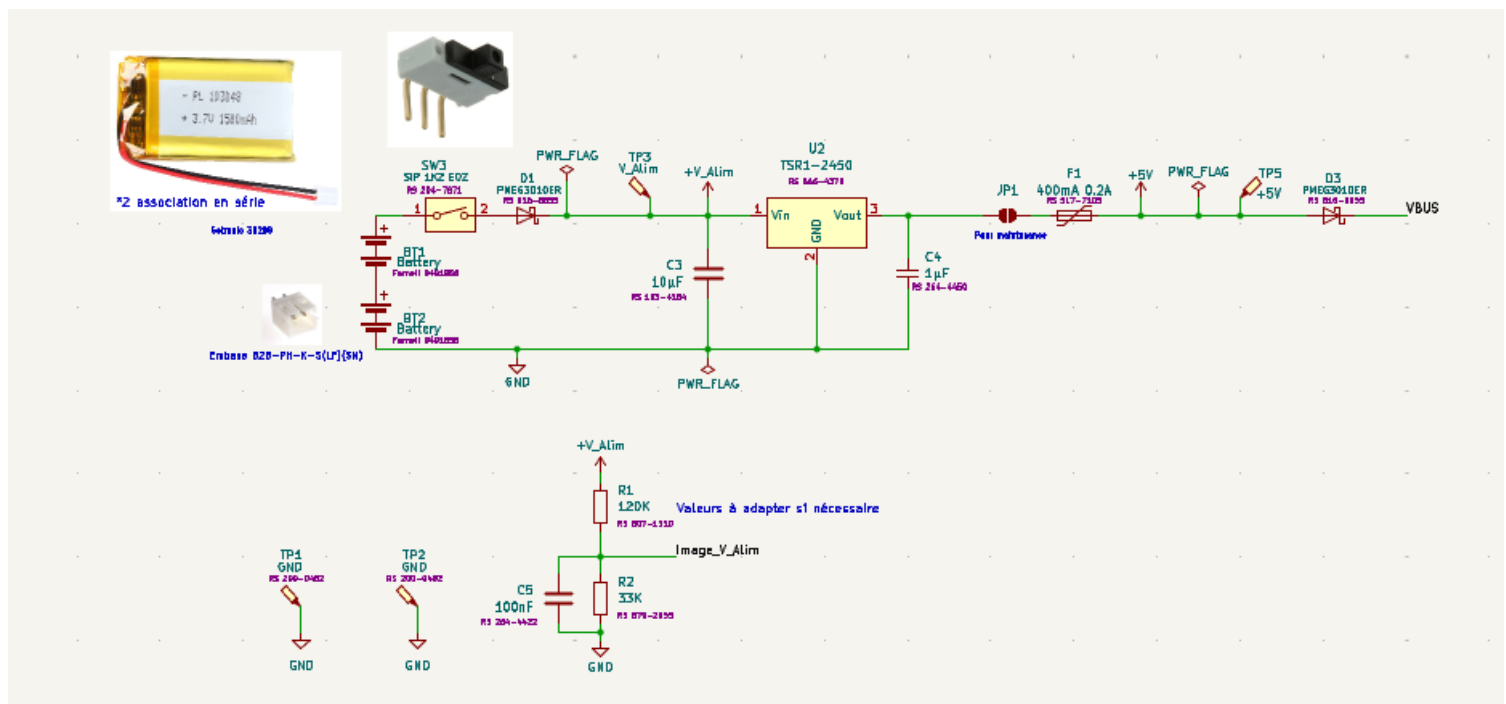
## Architecture interne:

### **Alimentation et Sécurité:**

Ce bloc gère l'énergie de la carabine. J'ai utilisé deux batteries LiPo montées en série pour obtenir une tension stable. La sécurité est assurée par une diode D1 qui protège l'électronique contre toute inversion de polarité lors du branchement, et un fusible réarmable F1 qui coupe le circuit en cas de court-circuit.

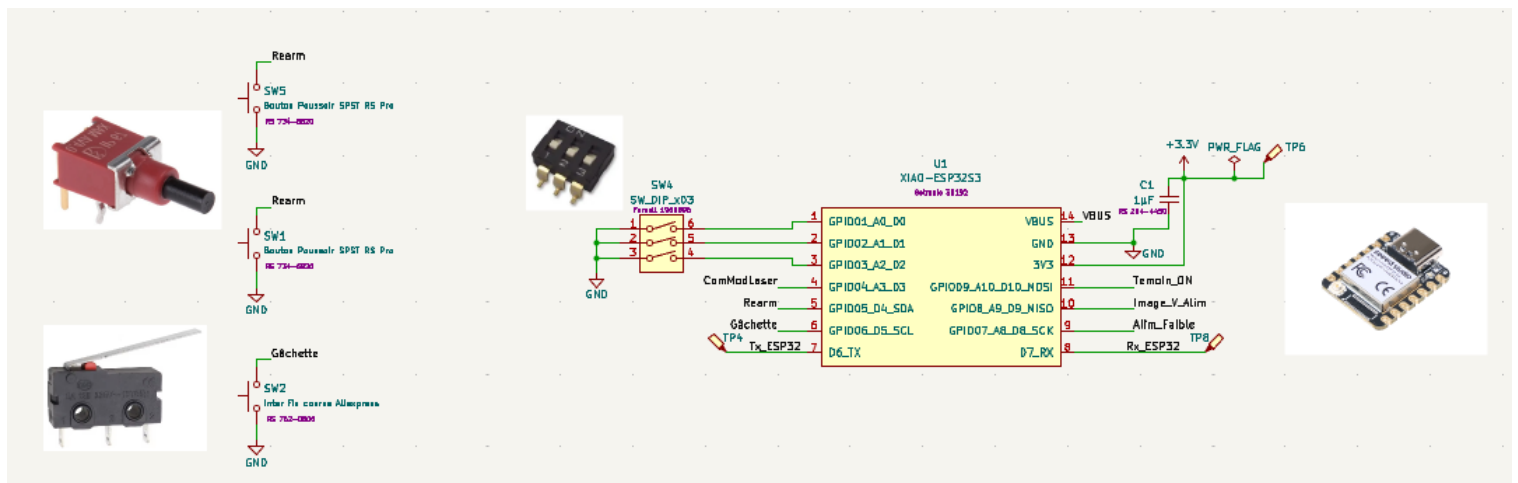
Pour fournir une tension propre aux composants sensibles, j'ai intégré un régulateur TSR1-2450. Il convertit la tension variable des batteries en un 5V constant. Enfin, un pont diviseur de tension (R1/R2) réduit la tension de la batterie pour que l'ESP32 puisse mesurer son niveau de charge sans risquer de griller son entrée analogique.

### bloc alimentation:



## BLOC 2 : Unité de traitement et Contrôle:

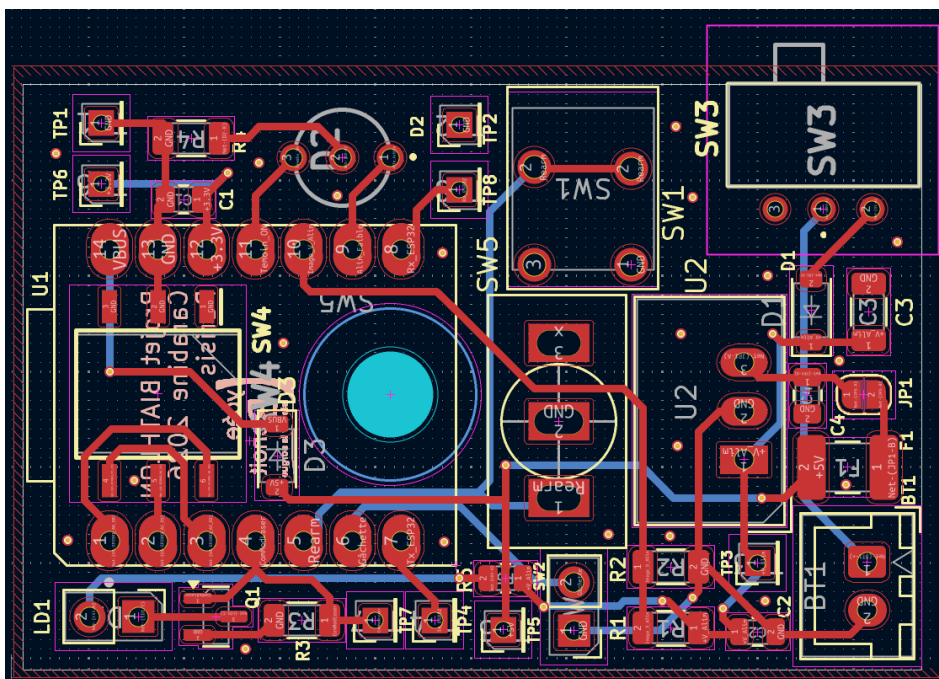
Le cœur du système est le module XIAO ESP32-S3. Ce microcontrôleur centralise toutes les informations. J'ai configuré les broches à l'aide des mes camarades pour gérer les différentes entrées utilisateur : la gâchette (SW2) pour le tir, les boutons de réarmement, et un sélecteur DIP Switch pour paramétrer les modes de fonctionnement. L'ESP analyse l'appui sur la gâchette, vérifie le niveau de batterie via le circuit de mesure, et envoie les ordres de commande aux sorties.



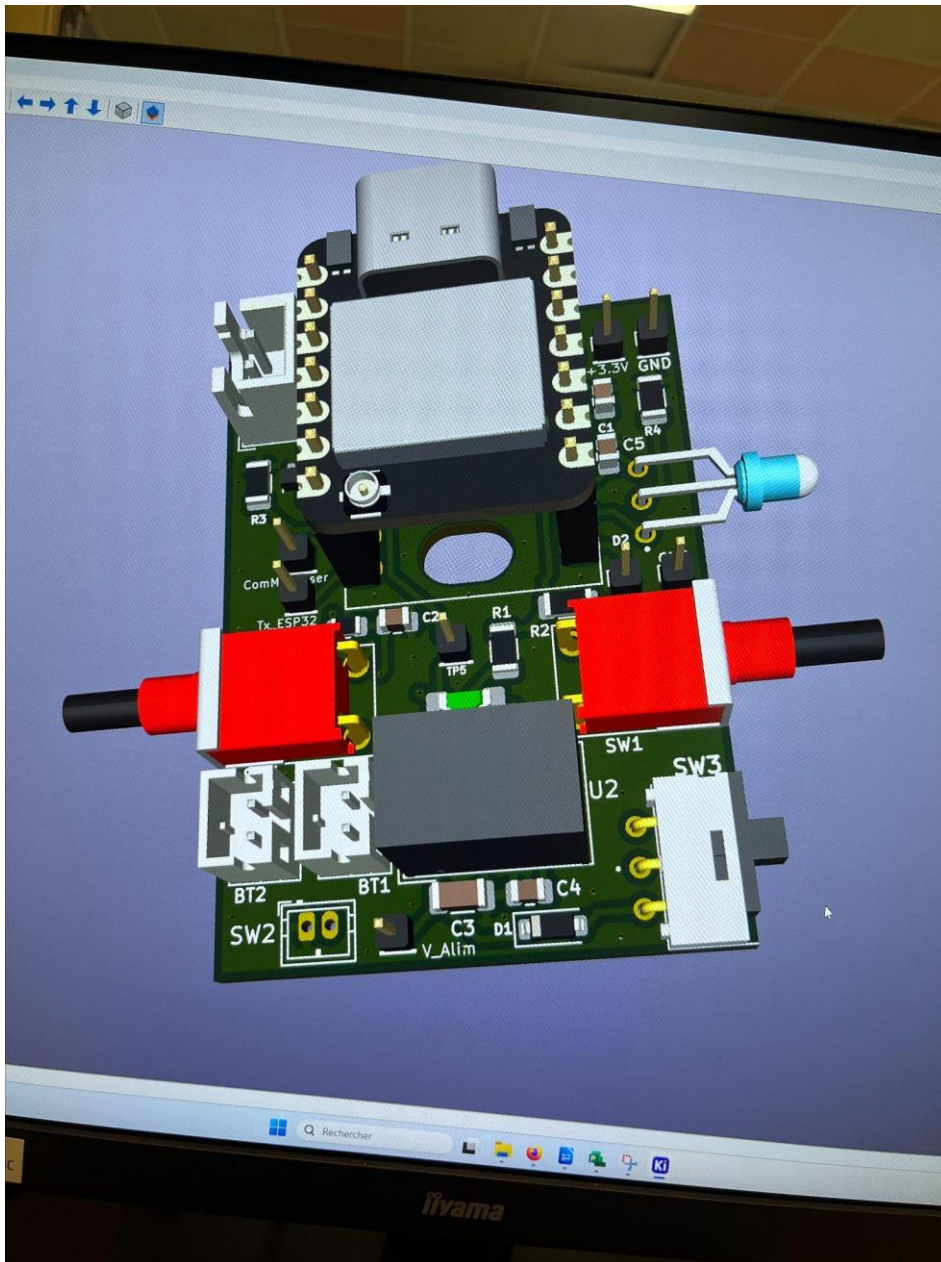
## BLOC 3 : Commande Laser et Interface

Ce bloc assure l'exécution du tir et le retour d'information. Comme l'ESP32 ne peut pas alimenter directement le laser, j'ai intégré un transistor MOSFET (Q1) qui agit comme un interrupteur de puissance. Quand le microcontrôleur reçoit l'ordre de tir, il active le MOSFET pour laisser passer le courant vers le module laser avec une précision maximale.

L'interface utilisateur est complétée par une LED bicolore. Elle informe le tireur sur l'état de la carabine : elle reste verte pour confirmer le bon fonctionnement et passe au rouge si le circuit de mesure détecte une batterie faible. Ce montage permet une interaction simple et immédiate entre la machine et l'utilisateur.

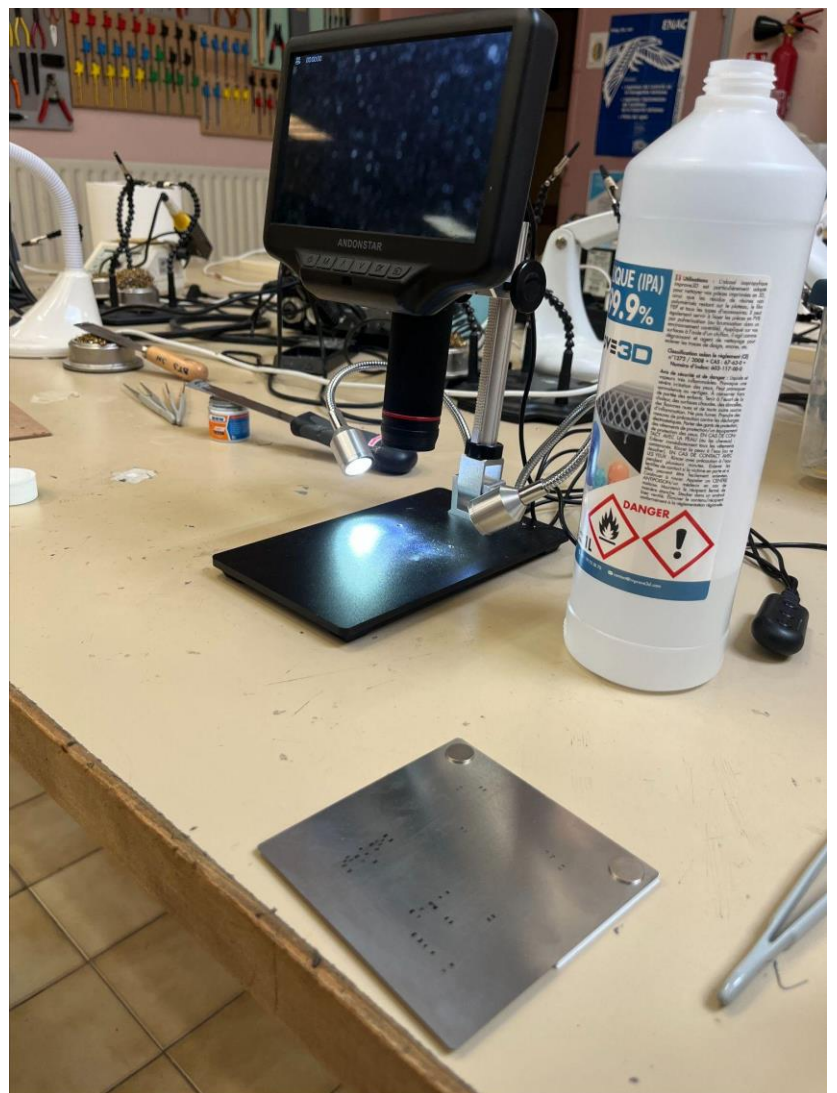


**Visualisation 3D (Rendu final de la carte) :**



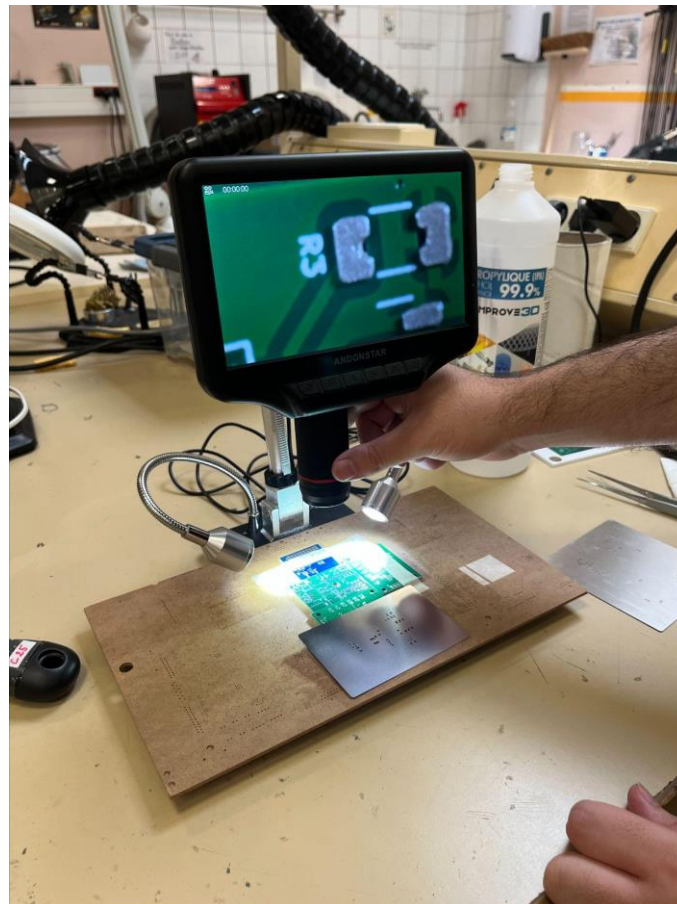
### **Processus de fabrication et d'assemblage:**

La fabrication de la carte a suivi un protocole rigoureux pour garantir la fiabilité des connexions. J'ai débuté par un nettoyage complet du PCB à l'alcool isopropylique afin d'éliminer toute impureté. Une fois la carte fixée sur son support, j'ai utilisé un stencil (pochoir) pour appliquer avec précision la pâte à braser sur les empreintes. Chaque dépôt a été minutieusement inspecté à la loupe pour valider l'uniformité du dosage avant la pose des composants.



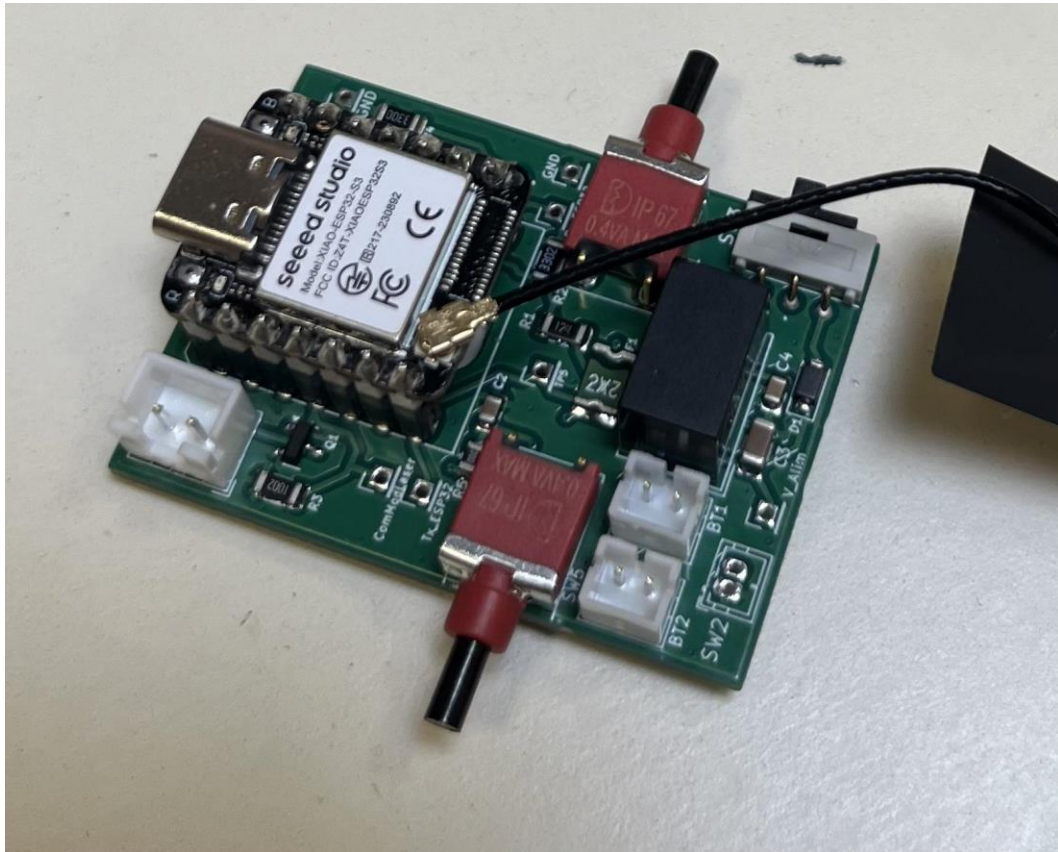
## Le montage s'est déroulé en deux phases :

1. **Phase CMS :** J'ai positionné les composants de surface (résistances, condensateurs etc...) en vérifiant scrupuleusement leur polarité si besoin. La carte a ensuite été placée dans un four à refusion pour fixer durablement ces composants par brasage thermique.
2. **Phase Traversante :** Le lendemain, j'ai finalisé l'assemblage manuellement au **fer à souder**. J'ai installé les composants traversants (connecteurs, interrupteurs) en respectant une progression logique, des plus petits aux plus volumineux, pour faciliter la manipulation et garantir la propreté des jointures d'étain.



**Carte finalisée:**





(A noter que nous avons mis 3 embrases différentes pour éviter à tout débutant de se tromper d'alimentation en essayant de brancher la connectique.)

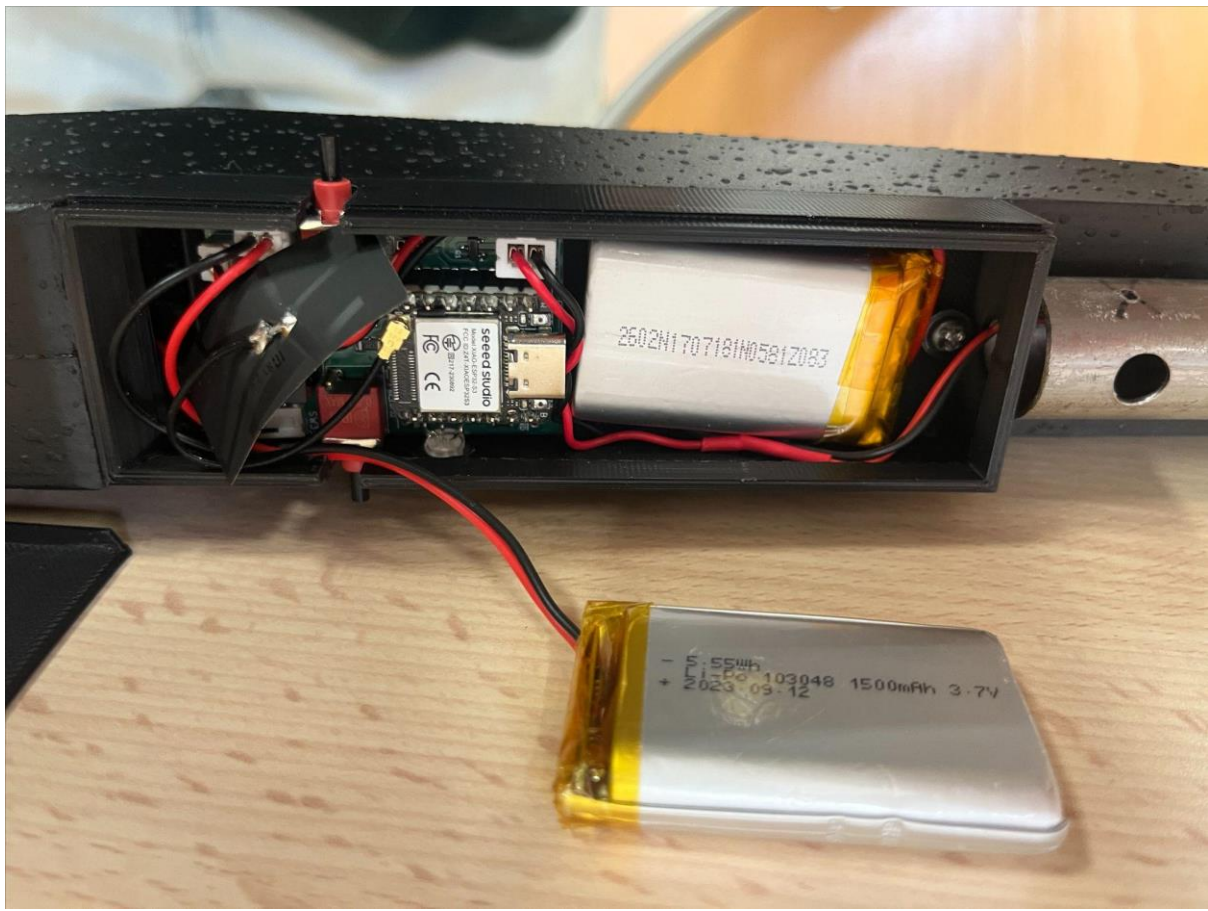
### Plans 3d de la carabine et version final:

Rendu d'une carabine laser modélisée



## Rendu de la carabine final avec carte intégrée





## Annexes

test communication avec télécommande :



Test de la borne wifi:

Code utilisé:

```
sketch_feb3a.ino
1 #include <Wifi.h>
2
3 // --- À MODIFIER ICI ---
4 const char* ssid = "BTS-SN-EC"; // Le nom exact de ta box
5 // -----
6
7 void setup() {
8 // Démarrage du port série
9 Serial.begin(115200);
10
11 // Pause pour laisser le temps à l'USB de s'activer (important pour le S3)
12 delay(2000);
13
14 Serial.println("\n--- Démarrage de la connexion ---");
15
16 // On lance la connexion
17 Wifi.mode(WIFI_STA); // Mode Station (comme un téléphone)
18 Wifi.begin(ssid);
19 Serial.print("Connexion en cours à ");
20 Serial.print(ssid);
21
22 // On attend tant qu'on n'est pas connecté
23 while (Wifi.status() != WL_CONNECTED) {
24 delay(500);
25 Serial.print(".");
26 }
27 }
```

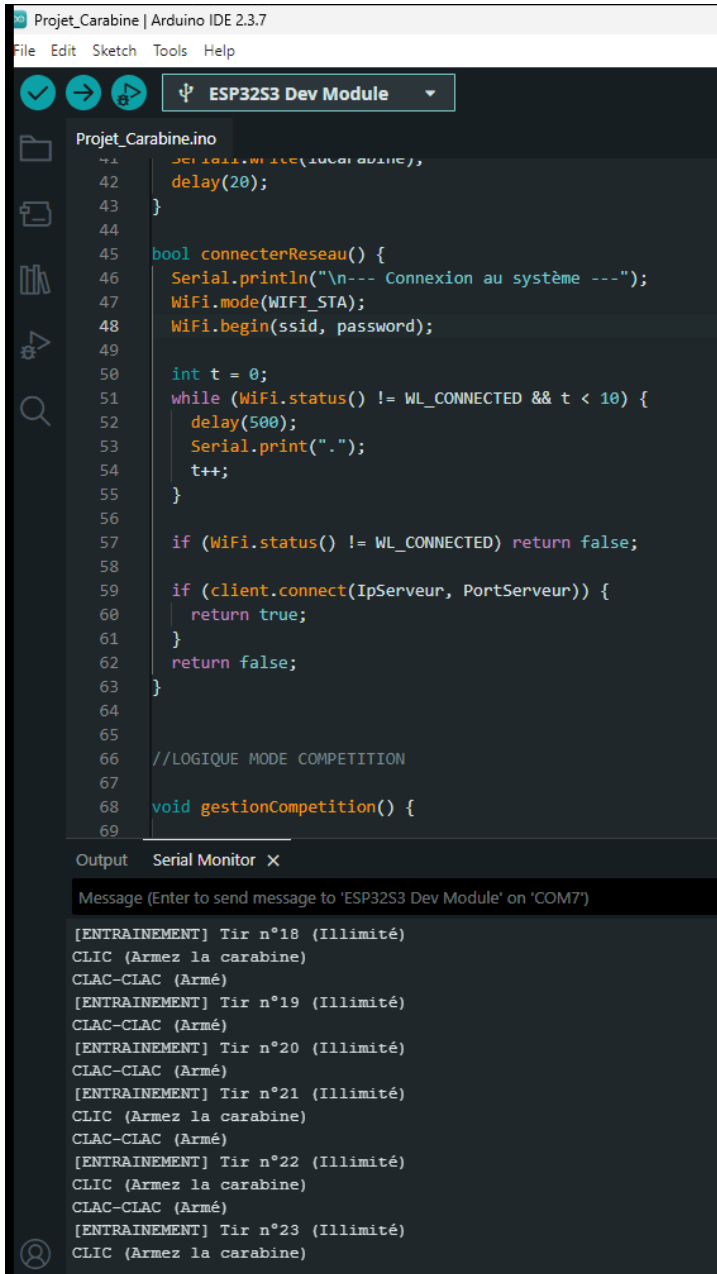
Output Serial Monitor X

Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM5')

```
--- Démarrage de la connexion ---
Connexion en cours à BTS-SN-EC.

--- C'EST CONNECTÉ ! ---
Adresse IP obtenue : 192.168.4.38
Force du signal (RSSI) : -85 dBm
```

Création code d'entraînement avec (un IR) devra cependant être modifié:



The screenshot shows the Arduino IDE interface. The top bar indicates the project name 'Projet\_Carabine' and the IDE version '2.3.7'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. The toolbar shows icons for checking, running, and uploading code, along with a dropdown menu for the board 'ESP32S3 Dev Module'. The left sidebar contains icons for file explorer, serial monitor, and search. The main editor displays the code for 'Projet\_Carabine.ino'. The code includes a delay, a network connection function, and a competition management function. The Serial Monitor at the bottom shows the output of the program, displaying training messages for 18, 19, 20, 21, 22, and 23 trials.

```
Projet_Carabine.ino
41 delay(1000); // Attente 1 seconde,
42 delay(20);
43 }
44
45 bool connecterReseau() {
46 Serial.println("\n--- Connexion au système ---");
47 WiFi.mode(WIFI_STA);
48 WiFi.begin(ssid, password);
49
50 int t = 0;
51 while (WiFi.status() != WL_CONNECTED && t < 10) {
52 delay(500);
53 Serial.print(".");
54 t++;
55 }
56
57 if (WiFi.status() != WL_CONNECTED) return false;
58
59 if (client.connect(IpServeur, PortServeur)) {
60 return true;
61 }
62 return false;
63 }
64
65
66 //LOGIQUE MODE COMPETITION
67
68 void gestionCompétition() {
69
```

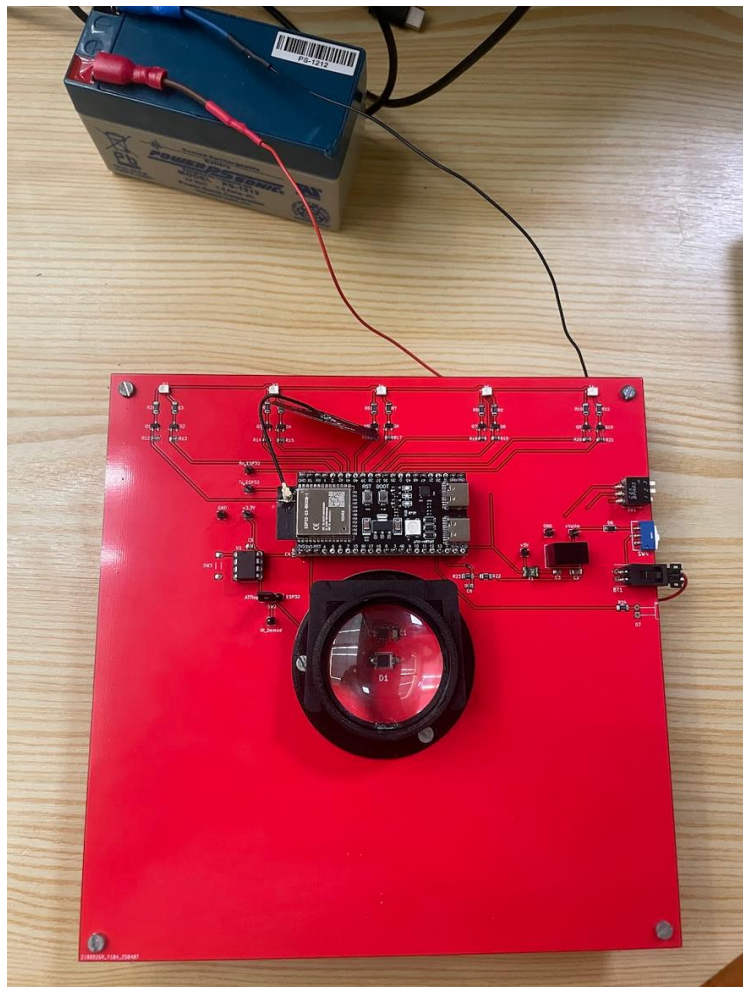
Output Serial Monitor X

Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM7')

```
[ENTRAINEMENT] Tir n°18 (Illimité)
CLIC (Armez la carabine)
CLAC-CLAC (Armé)
[ENTRAINEMENT] Tir n°19 (Illimité)
CLAC-CLAC (Armé)
[ENTRAINEMENT] Tir n°20 (Illimité)
CLAC-CLAC (Armé)
[ENTRAINEMENT] Tir n°21 (Illimité)
CLIC (Armez la carabine)
CLAC-CLAC (Armé)
[ENTRAINEMENT] Tir n°22 (Illimité)
CLIC (Armez la carabine)
CLAC-CLAC (Armé)
[ENTRAINEMENT] Tir n°23 (Illimité)
CLIC (Armez la carabine)
```

# **Timéo Klippel**

*Partie Personnelle- Timéo Klippel*



# SOMMAIRE:

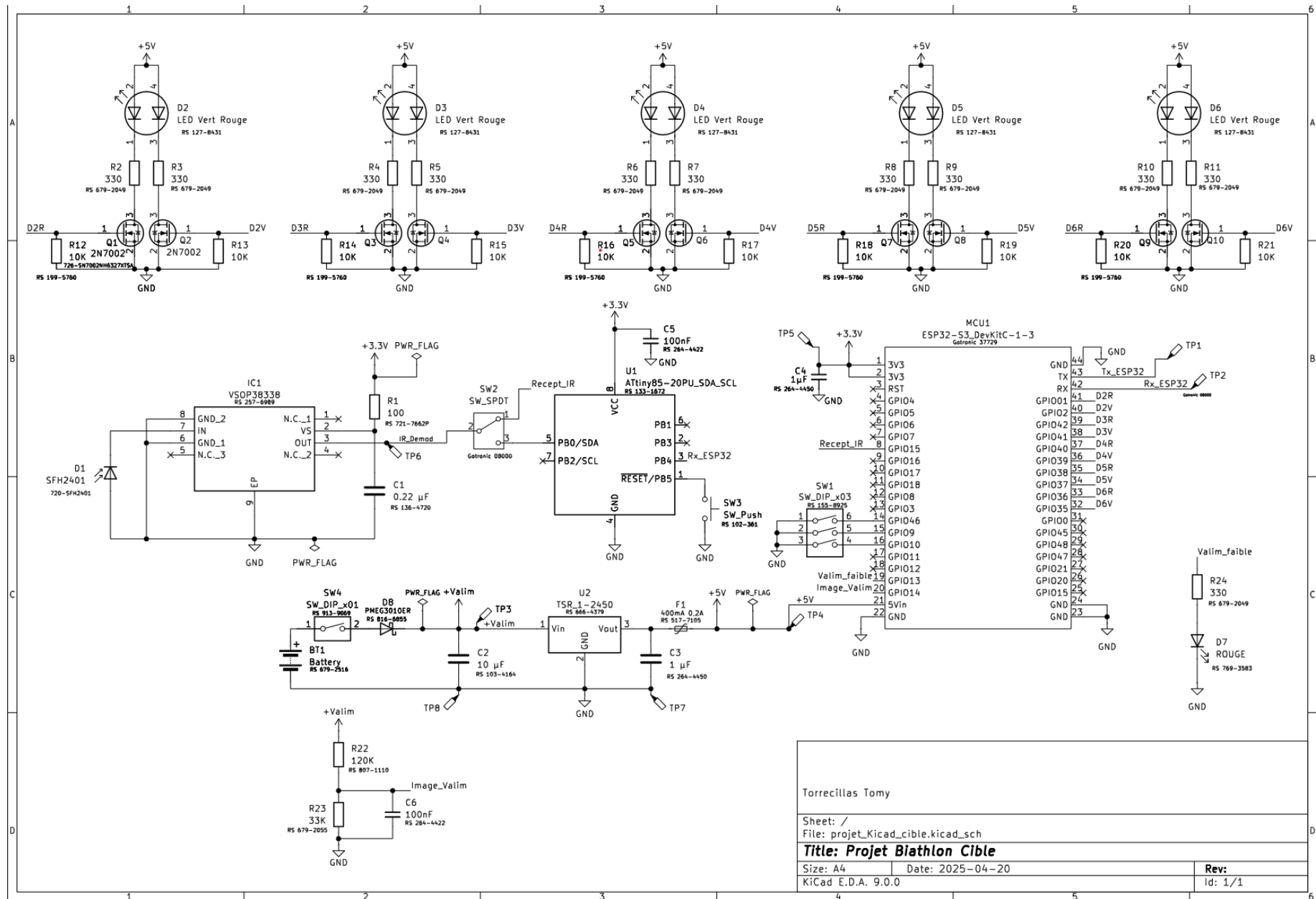
|                                                                                     |           |
|-------------------------------------------------------------------------------------|-----------|
| <b>1. Objectifs de la partie ( Cible Laser ).....</b>                               | <b>3</b>  |
| <b>2. Explication Globale du Fonctionnement de la Cible.....</b>                    | <b>5</b>  |
| <b>3. Planning prévisionnel pour les différentes tâches du projet :.....</b>        | <b>6</b>  |
| <b>4. Schéma Structurel pour 2026.....</b>                                          | <b>7</b>  |
| 4.1 Fonctionnement des LEDS Rouges et Vertes .....                                  | 8         |
| 4.2 Détection des tirs (Capteur SFH 4201).....                                      | 9         |
| 4.3 Microcontrôleur et son fonctionnement.....                                      | 10        |
| 4.4 Communication et contrôle:.....                                                 | 11        |
| <b>5. Déroulement par étapes de mon travail sur la cible. ....</b>                  | <b>12</b> |
| 5.1 Soudage de la carte: .....                                                      | 15        |
| 5.2 Suivi global de travail mensuel .....                                           | 16        |
| <b>6. Fiche de suivie personnelle .....</b>                                         | <b>20</b> |
| <b>7. Mise en Service de la Cible BIATHLON .....</b>                                | <b>21</b> |
| <b>8. Annexes :.....</b>                                                            | <b>23</b> |
| 8.1 Code de Test de la connexion WIFI :.....                                        | 23        |
| 8.2 Code de test des Leds :.....                                                    | 24        |
| 8.3 Utilisation de la station à air chaud pour la soudure des composants CMS: ..... | 25        |
| 8.4 Rendu de la carte après soudure des composants( Recto et Verso) : .....         | 26        |

## 1. Objectifs de la partie ( Cible Laser )

L'objectif principal est de concevoir la nouvelle architecture matérielle de la cible. Étant donné que c'est un projet reconduit des années précédentes, on parle ici d'une nouvelle version de la carte Cible version 2026. Mes missions incluent :

- **Optimisation Hardware** : Analyser la version 2025 pour simplifier le schéma, réduire l'encombrement, faciliter l'utilisation et donner un aspect plus fini et professionnel à la cible.
- **Authentication** : L'authentification et l'appairage d'une cible à une carabine se fait grâce aux switchs dédiés. Une idée première était de rajouter un circuit d'authentification (type ATECC608) pour sécuriser l'identité de la cible afin qu'une carabine falsifiée ou tout autre type de réplique ne venant pas du même constructeur soit évité. Cependant cette option a été abandonnée car trop dure à mettre en œuvre ici.
- **Conception Assistée par Ordinateur (CAO)** : Réaliser le schéma structurel et le routage (PCB) sous KiCad en coordination avec les étudiants IR pour les problèmes de mise en situation.
- **Fiabilisation de la réception** : Assurer la qualité du décodage des tirs laser via le ATtiny85 et filtrer les perturbations afin que le rendu soit satisfaisant.

**Cette analyse est accompagnée du schéma créé par l'année passée :**



Torrecillas Tomy

Sheet: /  
File: projet\_Kicad\_cible.kicad\_sch

**Title: Projet Biathlon Cible**

Size: A4 Date: 2025-04-20

KiCad E.D.A. 9.0.0

Rev:  
Id: 1/1

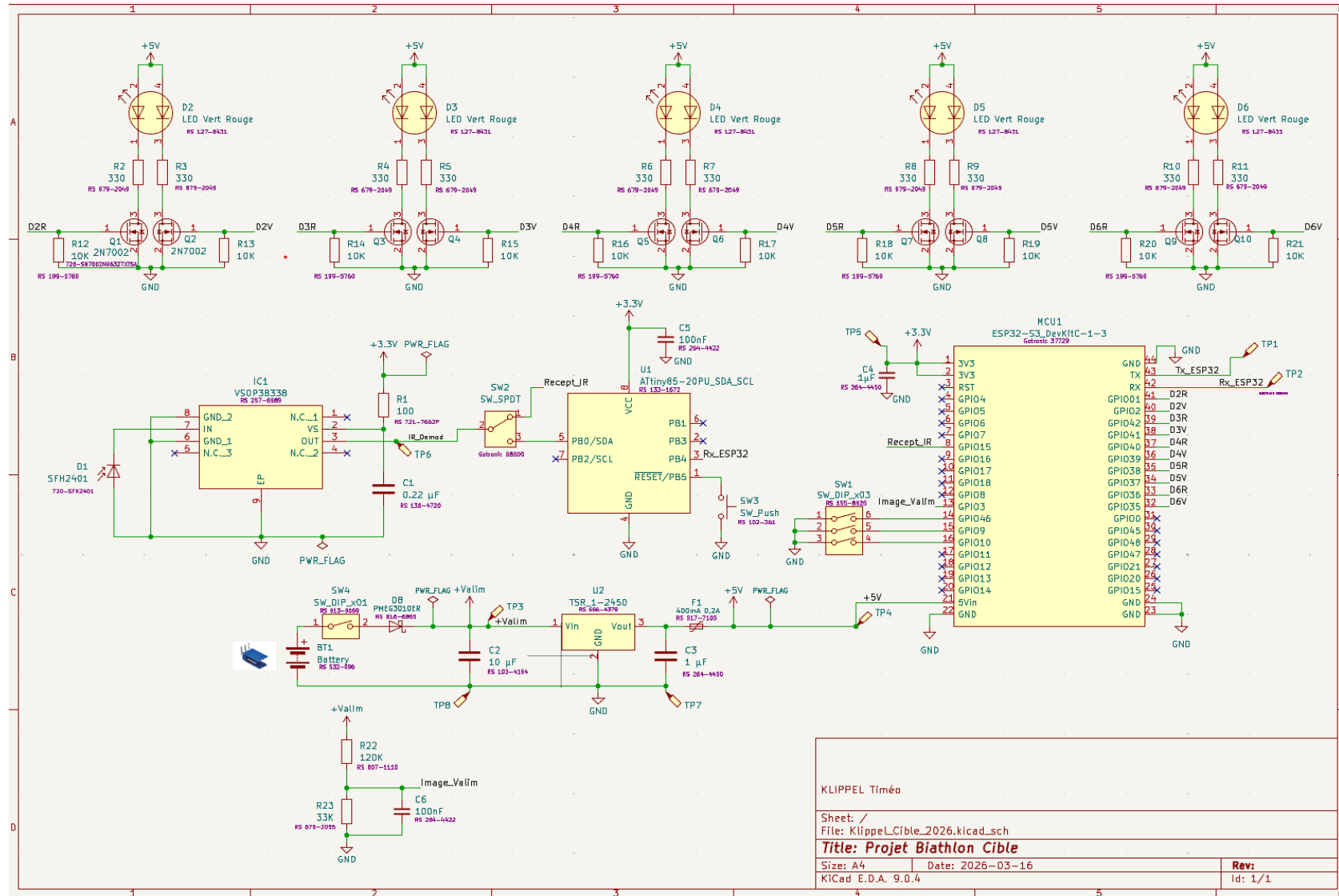
## **2. Explication Globale du Fonctionnement de la Cible**

- D'abord, il faut alimenter la cible. On utilise pour cela la batterie grâce au connecteur situé sur le bord de la carte.
- Une fois la cible alimentée, il faut appairer cette cible avec l'une des carabines grâce aux switchs situés sur le côté de la cible.
- Ensuite, le superviseur peut choisir de lancer le mode entraînement ou le mode compétition. Si aucune connexion WiFi n'est détectée, le mode entraînement se lance par défaut. Il y a également un allumage des 5 LEDS indiquant le niveau d'autonomie de la batterie au démarrage.

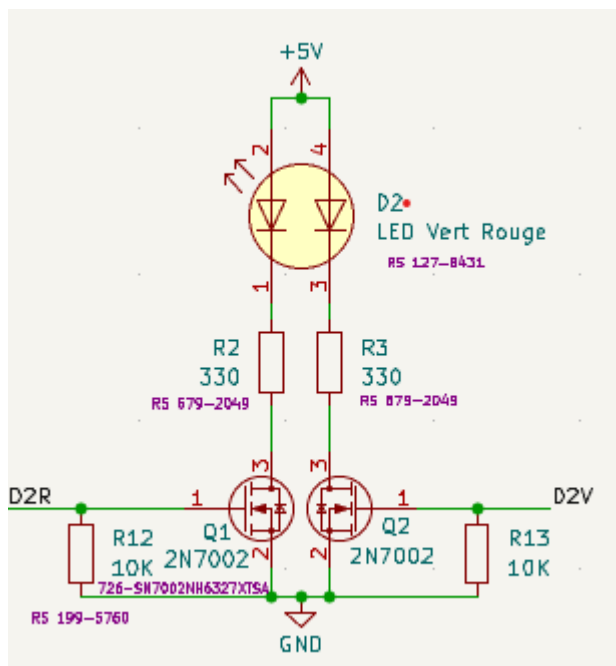
### 3. Planning prévisionnel pour les différentes tâches du projet :

|                                                             |       |              |              |
|-------------------------------------------------------------|-------|--------------|--------------|
| ▣ Évaluation du projet                                      | 84 h  | Mar 06/01/2  | Mar 07/04/2  |
| Revue 1                                                     | 1 h   | Mar 06/01/2  | Mar 06/01/2  |
| Revue 2                                                     | 1 h   | Mar 10/02/2  | Mar 10/02/2  |
| Revue 3                                                     | 1 h   | Mar 07/04/2  | Mar 07/04/2  |
| ▣ Projet                                                    | 12 h  | Mar 06/01/2  | Mar 13/01/2  |
| ▣ Prise en main commune                                     | 12 h  | Mar 06/01/26 | Mar 13/01/26 |
| Découverte du projet et de l'ancien dossier                 | 4 h   | Mar 06/01/26 | Mar 06/01/26 |
| Création du planning visé                                   | 4 h   | Lun 12/01/26 | Lun 12/01/26 |
| Approfondisse de l'étude du                                 | 4 h   | Mar 13/01/26 | Mar 13/01/26 |
| ▣ Partie ER 2                                               | 128 h | Lun 19/01/2  | Mar 16/06/2  |
| Analyse et tests des leds                                   | 12 h  | Lun 19/01/26 | Lun 26/01/26 |
| Mise en place du WiFi                                       | 20 h  | Lun 02/02/26 | Mar 03/03/26 |
| Modifications structurelles et optimisation de cette carte  | 20 h  | Mar 10/03/26 | Mar 24/03/26 |
| Essais de l'alimentation dans des conditions proche du réel | 16 h  | Lun 30/03/26 | Mar 28/04/26 |
| Saisie de schéma et Routage de la nouvelle carte            | 20 h  | Lun 04/05/26 | Lun 18/05/26 |
| Câblage de la carte et essais                               | 20 h  | Mar 19/05/26 | Lun 08/06/26 |
| Documentation de la mise en service et maintenance          | 10 h  | Mar 09/06/26 | Mar 16/06/26 |

## 4. Schéma Structurel pour 2026



## 4.1 Fonctionnement des LEDS Rouges et Vertes



Comme l'illustre cet extrait du schéma structurel, chaque LED bicolore est accompagnée de deux transistors. Ainsi, on peut piloter chacune d'entre elles individuellement et choisir grâce à ces transistors d'allumer soit le rouge, soit le vert, soit les deux simultanément afin de faire apparaître du orange.

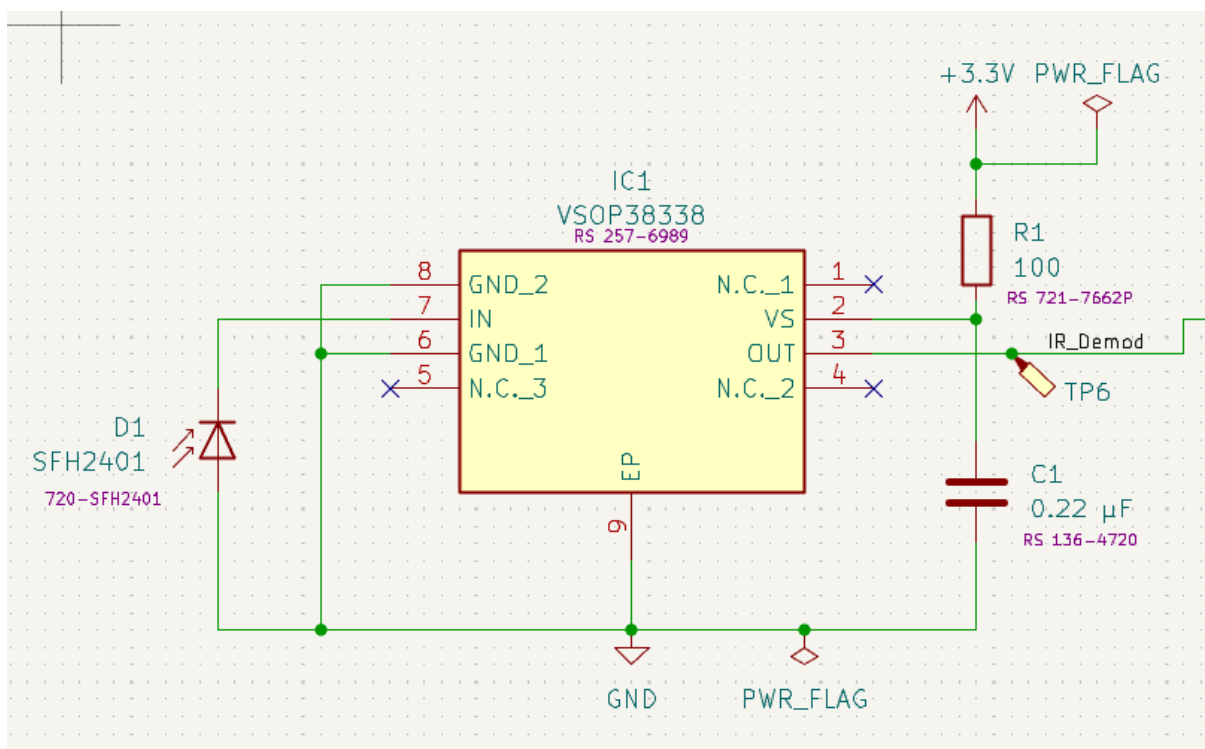
Dans notre cas, ces différents éclairages ont chacun des significations différentes : Si la LED n°1 s'allume en vert, alors le tir n°1 est réussi. A l'inverse, si elle est rouge, il est loupé. L'éclairage orange, quant à lui, sera utilisé pour signaler un niveau de batterie faible.

Ainsi, lorsqu'une phase de tir est en cours - Ici l'ordre de priorité des actions commandées par le code entre en jeu - les 5 LEDS s'allument afin que l'arbitre (ou le professeur) puisse remarquer le niveau de batterie et agir en conséquence.

Le fait d'indiquer le niveau de batterie avec les 5 LEDs déjà disponibles nous a d'ailleurs permis de supprimer la LED rouge (appelée D7 en bas à droite du schéma de 2025) qui était prévue à cet effet.

- Transistors MOS Canal N (Q2 à Q6 - 2N7002) : Ils agissent comme des interrupteurs électroniques permettant d'allumer ou d'éteindre les LEDs selon le signal de contrôle venant de l'ESP32
- Résistances associées (R12 à R21) : Elles limitent le courant pour protéger les LEDs et assurer un fonctionnement correct des transistors.

## 4.2 Détection des tirs (Capteur SFH 4201)



- Photodiode infrarouge (D1 - SFH 2401) : Cette photodiode détecte la présence d'un signal laser (ici un tir).
- Circuit amplificateur IC1 (VSOP38338) : un démodulateur pour photodiode, il permet de supprimer la porteuse à 38 KHz du signal infrarouge reçu et ainsi permettre la "traduction" de l'information.
- Résistances et condensateurs associés (R1, C1) : Ils stabilisent le signal et ajustent la sensibilité du capteur.

### 4.3 Microcontrôleur et son fonctionnement

#### - **ESP32-S3 (MCU1) :**

C'est le cerveau du système, il reçoit les signaux du capteur, traite les informations et commande les LEDs. Le choix de l'ESP 32 est dû à différentes choses :

- sa programmation peut se faire dans l'environnement de programmation Arduino, qui propose plein de librairies,
- la carte DEV Kit C1 donne accès à de nombreux GPIO du microcontrôleur,
- cette carte est facilement disponible chez les distributeurs.
- son prix est abordable,

- **Connexions GPIO :**

Utilisés pour interagir avec les LEDs et les capteurs.

- **Alimentation 3.3V et GND :**

L'ESP32 fonctionne sous 3.3V, fourni par un régulateur.  
La carte de développement doit être alimentée sous +5V ,  
elle génère du 3.3V , entre autres pour son  
microcontrôleur

#### **4.4 Communication et contrôle:**

- **Microcontrôleur ATtiny85 (U1) :** Sur la carte cible, l'ATtiny85 utilise la librairie IRremote pour lire les commandes envoyées en IR via le protocole NEC par la carabine laser (Ici rien de nouveau par rapport à l'année précédente) . Il transmet ensuite la donnée en UART à l'ESP32, qui, bien que capable de le faire, est déchargé de cette tâche pour se concentrer sur la gestion de la communication Wi-Fi avec le Superviseur par exemple.

- **Connexions PBO :** Elles permettent à l'ATtiny85 de communiquer avec l'ESP32 en UART pour transmettre les données décodées.

- **Bouton Reset (SW1) :** Permet de réinitialiser le microcontrôleur si nécessaire.

- **Batterie (BT1) :** Source d'énergie principale du système.

- **Régulateur de tension (C2, C3, SW4, etc.)** : Permet de stabiliser la tension d'alimentation pour les différents composants (ESP32 = 5V, LEDs = 5V ,ATtiny = 3.3V).

## 5. Déroulement par étapes de mon travail sur la cible.

- Lorsque le projet m'a été attribué, j'ai d'abord pris le temps d'étudier le travail effectué par les anciens étudiants afin de bien comprendre en premier lieu, le fonctionnement de la précédente carte, mais également la mise en œuvre général de l'épreuve avec les éléments déjà présents et les problèmes ainsi mis en avant dans les expérimentations passées.

A partir de cela j'ai pu en déduire les modifications nécessaires et les nouvelles fonctionnalités à intégrer dans la cible.

Pour cela , le cahier des charges à été mis à jour afin de conserver une ligne directrice lors du travail.

Le passage de la théorie à la pratique a révélé des points techniques importants :

- **Problème n°1 : Disponibilité et encombrement de l'ESP32.**

- **Symptôme** : On a remarqué des difficultés d'approvisionnement sur le modèle d'ESP 32 utilisé l'année précédente (**ESP32-S3 WROOM Dev-Kit C1**) et des risques de devoir redessiner entièrement l'empreinte de la carte si le besoin venait de trouver un nouveau modèle d'ESP32

- **Solution** : Validation après tests de la réutilisation du premier modèle d'ESP32 en y adaptant une pince métallique faisant office d'antenne Wi-Fi. Cela a pu garantir un état fonctionnel le temps de trouver une alternative plus professionnelle.
- Avec le temps nous avons pu trouver également un modèle d'antenne qui correspond à nos besoins :

## Antenne XIAO ESP32-S3 318020968



Ainsi qu'un connecteur femelle correspondant afin de pouvoir correctement utiliser cette antenne.

- **Problème n°2 : Instabilité de la réception laser.**

- **Symptôme** : La réception des tirs était parfois aléatoire ou déclenchée sans tir réel
- **Cause** : Nous avons d'abord pensé à une pollution lumineuse due aux néons de la salle de test créant des parasites sur le capteur IR. Cependant, la cause

la plus probable à priori était l'alimentation trop peu puissante du laser lors de nos essais car nous avons branché cette dernière grâce à un câble USB sur un poste de la salle.

- **Solution** : Analyse de la trame décodée par l'ATtiny85. Avec Lucas, nous avons repris le code de l'ATtiny pour ignorer les signaux non conformes et refait les tests avec la batterie prévue pour l'alimentation en autonomie de la cible lors de l'épreuve( Batterie 12V, 1.2Ah au plomb) Cette batterie offrant une meilleure alimentation, le laser était donc plus puissant et permit une bien meilleure réception du signal.

- **Problème n°3 : Validation Wifi.**

- **Symptôme** : Besoin de confirmer que le wifi fonctionne et que la portée est suffisante en situation réelle car la mise en œuvre se ferait naturellement en extérieur, ainsi certains paramètres comme la lumière du soleil ou les obstacles du terrain pourraient affecter la réception selon l'endroit. De plus, la pince servant d'antenne temporaire avait émis un premier doute sur la portée du signal qui s'annonçait à peine juste.
- **Solution** : Réalisation d'un programme de tests sur l'environnement Arduino pour valider la connexion Wifi (VOIR ANNEXE 1), assurant la cohérence avec le code de mon collègue (Lucas). Nous avons d'abord effectué les tests en salle de travail puis

nous nous sommes rendus sur le terrain qui accueillera l'épreuve. Épaulés par M.Leduc, nous avons pu faire nos tests en mettant en œuvre la nouvelle antenne sur les distances exactes et le lieu précis afin de garantir que la portée sera suffisante.

## **5.1 Soudage de la carte:**

### **1.Nettoyage :**

On nettoie la carte PCB avec de l'alcool isopropylique pour écartier toute poussière ou autre débris afin d'avoir une base saine puis on la fixe sur un support avec du ruban double face pour éviter que la carte ne bouge .

### **2.Positionner le stencil :**

On aligne soigneusement un stencil sur mesure en inox qui nous a été fourni par le distributeur lors de la conception du PCB à partir des fichiers de construction, on s'assure que les ouvertures correspondent exactement aux pads CMS lorsqu'on le place sur la carte.

### **3.Application de la pâte à braser :**

On met une petite quantité de pâte à braser sur une raclette puis on étale uniformément sur les ouvertures tout en appuyant fermement le stencil afin qu'il soit plaqué contre la carte et qu'il n'y ait pas de débordement, puis on retire le stencil avec précaution.

#### 4.Placement des composants CMS :

Utilisation d'une pince brucelle pour déposer les composants sur la pâte , on aligne les composants le plus précisément possible même si la pâte corrigera légèrement le placement au moment de la soudure à l'air chaud. Il faut également faire attention à orienter les composants de façon à ce que les inscriptions visibles dessus soient lisibles dans le même sens.

#### 5.Chauffage des composants :

On utilise normalement un four à refusion pour cette partie mais ici ma carte est bien trop grande nous avons alors opté pour une station à air chaud et avons chauffé progressivement en balayant chaque zone uniformément afin de ne pas dégrader les composants et circuits imprimés comme les diodes qui sont en partie en plastique.

#### 6.Inspection :

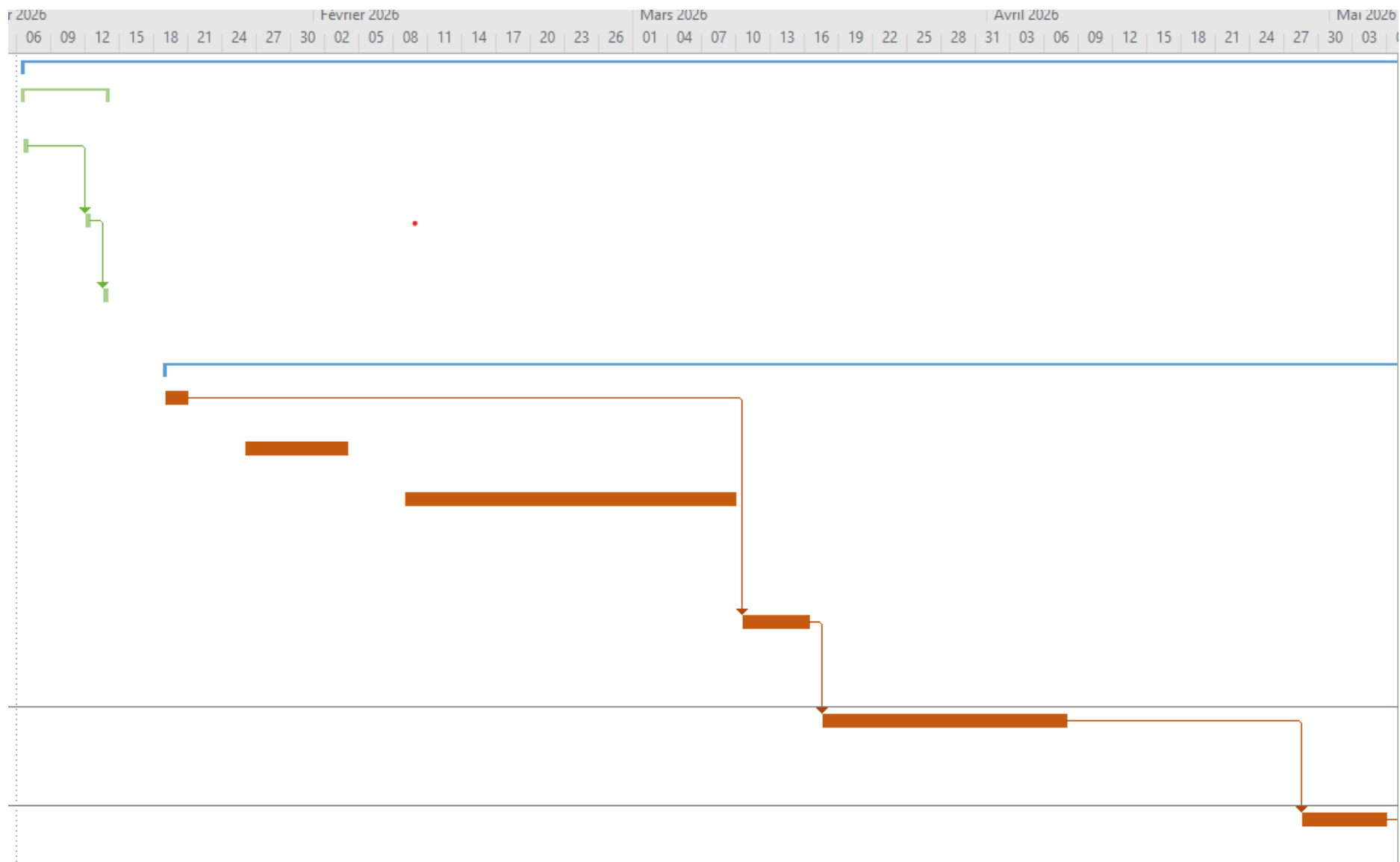
On vérifie ensuite à l'œil nu ou à la loupe les soudures et reprend manuellement à la panne si nécessaire. Ici, sur la zone de réception du laser, les plus petits composants ont eu plus de mal à se souder à l'air chaud donc j'ai retravaillé cette partie au fer à souder

### **Rendu de la carte après soudure recto et verso en ANNEXES**

## 5.2 Suivi global de travail mensuel

Mon avancement est conforme au planning prévisionnel finalisé le 13 janvier comme le montre ce diagramme de Gantt qui est correspondant à l'avancement actuel. A présent, la principale étape manquante sera de documenter la mise en service et la maintenance primaire de la cible.

|                                                             |              |                     |                     |
|-------------------------------------------------------------|--------------|---------------------|---------------------|
| ▲ <b>Projet</b>                                             | <b>140 h</b> | <b>Mar 06/01/26</b> | <b>Mar 16/06/26</b> |
| ▲ <b>Prise en main commune</b>                              | <b>12 h</b>  | <b>Mar 06/01/26</b> | <b>Mar 13/01/26</b> |
| Découverte du projet et de l'ancien dossier                 | 4 h          | Mar 06/01/26        | Mar 06/01/26        |
| Création du planning prévisionnel                           | 4 h          | Lun 12/01/26        | Lun 12/01/26        |
| Approfondisse de l'étude du projet                          | 4 h          | Mar 13/01/26        | Mar 13/01/26        |
| ▲ <b>Partie ER 2</b>                                        | <b>128 h</b> | <b>Lun 19/01/26</b> | <b>Mar 16/06/26</b> |
| Analyse et tests des leds                                   | 8 h          | Lun 19/01/26        | Mar 20/01/26        |
| Mise en place du WiFi                                       | 16 h         | Lun 26/01/26        | Mar 03/02/26        |
| Essais de l'alimentation dans des conditions proche du réel | 16 h         | Lun 09/02/26        | Lun 09/03/26        |
| Modifications structurelles et optimisation de cette carte  | 8 h          | Mar 10/03/26        | Lun 16/03/26        |
| Saisie de schéma et Routage de la nouvelle carte            | 24 h         | Mar 17/03/26        | Mar 07/04/26        |
| Câblage de la carte et essais                               | 12 h         | Mar 28/04/26        | Mar 05/05/26        |
| Documentation de la mise en service et maintenance          | 16 h         | Lun 08/06/26        | Mar 16/06/26        |



- **Janvier** : J'ai pu m'approprier le cahier des charges et analyser de manière à comparer le schéma 2025, j'ai pu noter les principaux changements à effectuer comme une dérivation faite sur mesure l'année dernière à mettre en œuvre de manière propre cette année.
- **Février** : Phases de tests unitaires (LEDs, Wi-Fi, Portée) et validation des choix technologiques. Nous avons eu l'opportunité de discuter plusieurs fois avec M.Leduc, le professeur d'EPS à l'origine de tout ce projet afin de voir avec lui si des modifications ou améliorations étaient à apporter selon son point de vue et sa vision de l'épreuve.
- **Mars** : Dédié à la saisie de schéma, et au routage afin que la commande du PCB puisse être effectuée avant les vacances d'avril. J'ai également le temps de travailler sur les démonstrations prévues dans la présentation finale et la revue globale de ma partie de projet.
- **Avril / Mai** : La saisie du schéma et le routage de la carte sont finis en respectant les nouvelles modifications, on projette de tester cette nouvelle carte et de produire une fiche d'utilisation et de maintenance pour clôturer ce projet.

## 6. Fiche de suivie personnelle

- Mar 06/01 : Découverte du projet ( cahier des charges ) + mini formation Project
- Lun 12/01 : Prise en main de Project et compréhension plus approfondie du projet Diagramme de Gantt commencé.
- Mar 13/01 : Présentation de projet avec les professeurs (M.Escuret) puis diagramme de Gantt terminé.
- Lun 19/01 : Analyse du schéma structurel de 2025 et de la cible pour comprendre le rôle de chacun des composants
- Mar 20/01 :Après quelques tests, on remarque qu'il est possible de réutiliser le premier ESP-32 en y ajoutant une petite pince métallique qui fera office d'antenne wifi. Cela permet d'éviter la redimension et surtout, ce composant est beaucoup plus disponible sur le marché. Tests d'allumages des leds OK POUR TOUTES LES COULEURS ET TOUTES LES LEDS
- Lun 26/01 : Prise en main de fritzing et des manipulations autour de L'ATTiny. Test du code de mise en fonctionnement minimal de la cible.
- Mar 27/01: Analyse du système Wifi
- Lun 02/02: création du programme de test wifi NON FINI
- Mar 03/02: finalisation du programme de test Wifi fonctionnel reste à tester la portée.
- Lun 09/02: Tests de la réception avec le code de L'ATTINY. On se rends compte qu'il arrive que la réception soit perturbée, sans doute à cause des néons de la salle
- Mar 10/02: Test pour voir si l'ATTiny décode bien la trame.
- Lun 09/03: schema kicad et diapo de revue + démo tests leds et wifi

- Mar 10/03: Revue de projet N°2
- Lun 16/03: Base de schéma terminées, voir pour modifier par rapport à l'année dernière (Voir pour disposer le circuit autrement avec la nouvelle antenne) + tests portée nouvelle antenne en extérieur !!! Ping échoue, à réessayer + modifier aussi sur le schéma le raccord fait sur la broche GPIO3
- Mar 17/03: Reprise du schéma et du pcb suites aux commentaires faits après partage de la veille, notamment les empreintes à mettre à jour.(routage fini) + travail revue 2
- Lun 23/03: Routage et empreintes finies, focalisation sur la revue car bcp de choses à améliorer (A noter le changement du connecteur d'alimentation de la carte pour qu'il soit plus pratique.

## 7. Mise en Service de la Cible BIATHLON

### Matériel Nécessaire :

- Carte cible câblée au préalable
- Carabine fonctionnelle
- ATtiny85 + programmeur Sparkfun
- ESP32-S3 DEVKIT
- Alimentation : Batterie ou alimentation de labo. (12V)
- Multimètre

### Procédure de Mise en Service :

- Avant de placer les circuits imprimés, utiliser un multimètre en position test diode pour vérifier que les leds rouges et vertes de D2 sont fonctionnelles. Pour ce faire, présenter le + du multimètre au point de test +5V et le - à l'extrémité basse de R2 puis de R3. Faire de même avec D3, D4, D5 et D6.

- Utiliser une batterie 12V au plomb ou une alimentation de laboratoire afin de mettre la carte cible sous tension en la raccordant par le connecteur BT1. Ne pas oublier de mettre SW4 en position ON.
- Vérifier entre les points de tests +5V et GND qu'on à bien une tension de +5V grâce au multimètre.
- Mettre SW4 en position OFF.
- Si le ATtiny85 est neuf et n'a jamais été programmé, graver la séquence d'initialisation grâce au programmeur Sparkfun et à L'IDE Arduino.
- Programmer ensuite le ATtiny85 en utilisant le sketch prévu à cet effet, toujours sur l'IDE Arduino.
- Placer le ATtiny85 sur son support en veillant à le mettre dans le bon sens.
- Programmer ensuite L'ESP32 en le connectant par USB sur le port convenu à cet effet. Utiliser ensuite de la même manière le sketch adapté depuis l'environnement Arduino.
- Débrancher ensuite l'USB et placer L'ESP32 sur son support toujours en faisant attention au sens.
- Mettre la cible sous tension (SW4 sur ON).
- Disposer SW1 à l'identique de celui sur la carabine afin de lier cible et carabine.
- Viser et appuyer sur la détente, la première Led affichera le résultat.

## 8. Annexes :

### 8.1 Code de Test de la connexion WIFI :

```
1 #include <WiFi.h>
2
3 const char* ssid = "PMV-RPI";
4 const char* password = "pmv-2024";
5
6 void setup() {
7 // Démarrage du port série
8 Serial.begin(9600);
9
10
11 delay(2000);
12
13 Serial.println("\n--- Démarrage de la connexion ---");
14
15 // On lance la connexion
16 WiFi.mode(WIFI_STA);
17 WiFi.begin(ssid, password);
18 Serial.print("Connexion en cours à ");
19 Serial.print(ssid);
20
21 // On attend tant qu'on n'est pas connecté
22
23 }
24
25 void loop() {
26
27 // L'ESP32 reste connecté en tâche de fond
28 while (WiFi.status() != WL_CONNECTED) {
29 delay(500);
30 Serial.println(".");
31 Serial.println(WiFi.macAddress());
32 }
33 delay(1000);
34
35 Serial.println("\n\n--- C'EST CONNECTÉ ! ---");
36 Serial.print("Adresse IP obtenue : ");
37 Serial.println(WiFi.localIP());
38 Serial.print("Force du signal (RSSI) : ");
39 Serial.print(WiFi.RSSI());
40 Serial.println(" dBm");
41 }
```

## 8.2 Code de test des Leds :

essais\_leds\_cible.ino

```
1
2 const int led1RedPin = 1 ;
3 const int led1GreenPin = 2 ;
4 const int led2RedPin = 42;
5 const int led2GreenPin = 41 ;
6 const int led3RedPin = 40 ;
7 const int led3GreenPin = 39 ;
8 const int led4RedPin = 38 ;
9 const int led4GreenPin = 37 ;
10 const int led5RedPin = 36 ;
11 const int led5GreenPin = 35 ;
12
13
14 void setup() {
15 pinMode(led1RedPin, OUTPUT);
16 pinMode(led1GreenPin, OUTPUT);
17 pinMode(led2RedPin, OUTPUT);
18 pinMode(led2GreenPin, OUTPUT);
19 pinMode(led3RedPin, OUTPUT);
20 pinMode(led3GreenPin, OUTPUT);
21 pinMode(led4RedPin, OUTPUT);
22 pinMode(led4GreenPin, OUTPUT);
23 pinMode(led5RedPin, OUTPUT);
24 pinMode(led5GreenPin, OUTPUT);
25 }
26
27 void loop() {
28
29 digitalWrite(led1RedPin, HIGH);
30 delay(300);
31 digitalWrite(led1RedPin, LOW);
32 delay(300);
33 digitalWrite(led1GreenPin, HIGH);
34 delay(300);
35 digitalWrite(led1GreenPin, LOW);
36 delay(300);
37 digitalWrite(led1RedPin, HIGH);
38 digitalWrite(led1GreenPin, HIGH);
39 delay(300);
40 digitalWrite(led1RedPin, LOW);
41 digitalWrite(led1GreenPin, LOW);
42 delay(300);
43 }
```

### 8.3 Utilisation de la station à air chaud pour la soudure des composants CMS:



## 8.4 Rendu de la carte après soudure des composants( Recto et Verso) :

