

Année 2021

Rapport de projet - S4P GEII Projet de ruche connecté

“SysBee”

Projet réalisé par :

Antoine Serry

Raoul Petrean

Ihsan Decieux

Songchaiyavat Sengchan

Tuteur du projet :

Denis Pénard



Avant Propos :

Ce rapport s'inscrit dans l'obtention du diplôme de technicien supérieur. Nous avons assigné un projet selon nos préférences. Au cours de cette première moitié de quatrième semestre, nous avons mené à bien l'étude et la fabrication d'un système de détection et d'alerte en cas essaimage. En voici le résultat.

Abstract :

This report brings an end to our 4th semester project at the IUT. We were given subjects based on our personal affinities. This first half semester was mostly dedicated to working on its completion. This is the result of how we successfully conducted the design of a swarming detection and alert system.

Un peu de contexte :

Afin de constituer les équipes, on a donné nos préférences parmi plusieurs spécialités : Programmation, design électronique, énergie et mathématiques. Nous avons choisi le projet SysBee car c'est celui qui ferait intervenir le plus de notions apprises au cours de notre cursus. Cette diversité explique pourquoi nous étions le seul groupe de quatre. De plus, nous étions déjà familiers avec le concept de SysBee grâce à notre tuteur Denis Pénard, qui nous en avait parlé lors des itérations précédentes.

Étudiant auteur de cette partie : Antoine SERRY

Sommaire

Résumé / Abstract	3
Introduction	5
1. Adaptation matérielle à la ruche	6
1.1. Étude de l'enveloppe extérieure de la carte SysBee	
1.1.1. Étude de la structure générale de l'enveloppe	
1.1.2. Adaptation à l'environnement	
1.2. Étude des capteurs optiques et des couloirs	
1.3. Conception du plateau de test	
2. Etude du matériel hardware de la carte SysBee	18
2.1. Alimentation	
2.2. Capteurs optiques	
2.2.1. Alimentation	
2.2.2. Résistances de charge	
2.3. Multiplexage	
2.3.1. Pourquoi utiliser des multiplexeurs ?	
2.3.2. Comment les utilise-t-on ?	
2.3.3. Placement et routage	
2.4. Microcontrôleur et composants auxiliaires	
2.4.1. Choix du microcontrôleur	
2.4.2. Condensateurs de découplage	
2.4.3. Voyants lumineux	
2.5. Périphériques de communication	
2.6. Fabrication et test, modifications	
3. Programmation, fonctionnement du logiciel	25
3.1. Automate global	
3.2. Échantillonnage et comptage	
3.3. Signaux lumineux	
3.4. Interface UART, one wire et USB	
4. Communications	30
4.1. Programmation du Xbee par UART	
4.2. Envoi et réception de trames	
4.3. TTN et le protocole LoRaWAN	
4.4. Protocole MQTT	
4.5. Application Android	

Introduction

L'essaimage est un évènement primordial dans l'apiculture. Au printemps, en particulier au mois de mai, la moitié des abeilles d'une ruche s'échappe en quelques minutes. Cela représente souvent plus de trente mille abeilles. Cette division de la colonie a pour objectif d'en fonder d'autres et ainsi d'étendre la population. C'est une vraie aubaine pour l'apiculteur qui peut à cet instant capturer l'essaim non seulement pour fonder une nouvelle ruche mais surtout pour éviter une prolifération incontrôlée.

Bien que spectaculaire, l'essaimage est aussi imprévisible. L'éleveur a intérêt à surveiller son rucher lors de cette période, car on a vite fait de laisser l'essaim s'échapper. Cette scrutation est une activité chronophage mais nécessaire. L'idée de notre projet SysBee est de concevoir un système qui détecte l'essaimage et alerte l'apiculteur.

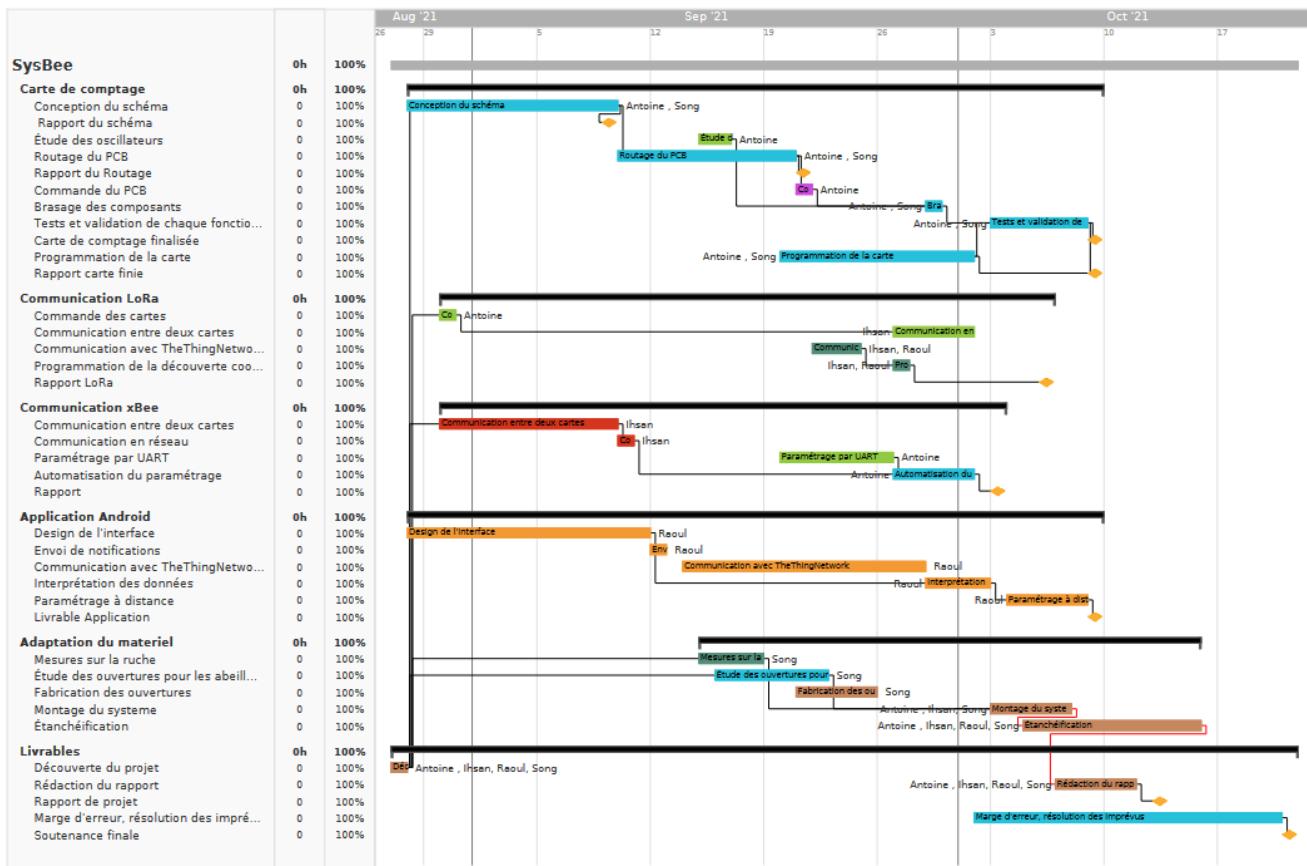
Il existe plusieurs moyens de le détecter. Nos prédécesseurs ont mesuré le poids des ruches avec des jauges de contrainte. Nous avons aussi imaginé avec Mr Pénard une reconnaissance du spectre audio spécifique au bruit d'un essaim. Pour cette année, nous avons décidé de mesurer les allées et venues des abeilles à l'aide de capteurs optiques. Nous nous sommes donc posé la problématique suivante:

Comment détecter un essaimage à l'aide de capteurs optiques, et alerter l'apiculteur ?

Pour y répondre nous avons découpé le travail en plusieurs axes principaux: La conception d'un appendice pour une ruche qui accueillerait notre système électronique, la conception du circuit électronique de ce système, sa programmation ainsi que la communication entre les ruches, internet, et l'apiculteur. Notre objectif est que toutes les ruches équipées puissent détecter un essaimage, et communiquer à courte portée avec les autres ruches. Une seule parmi elles a un module LoRa, lui permettant à terme de communiquer avec internet, et le smartphone de l'apiculteur.

Étudiant auteur de cette partie : Antoine SERRY

0 - Diagramme de Gantt du projet SysBee



Étudiant auteur de cette partie : Songchaiyavat SENGCHAN

1 – Adaptation matérielle à la ruche

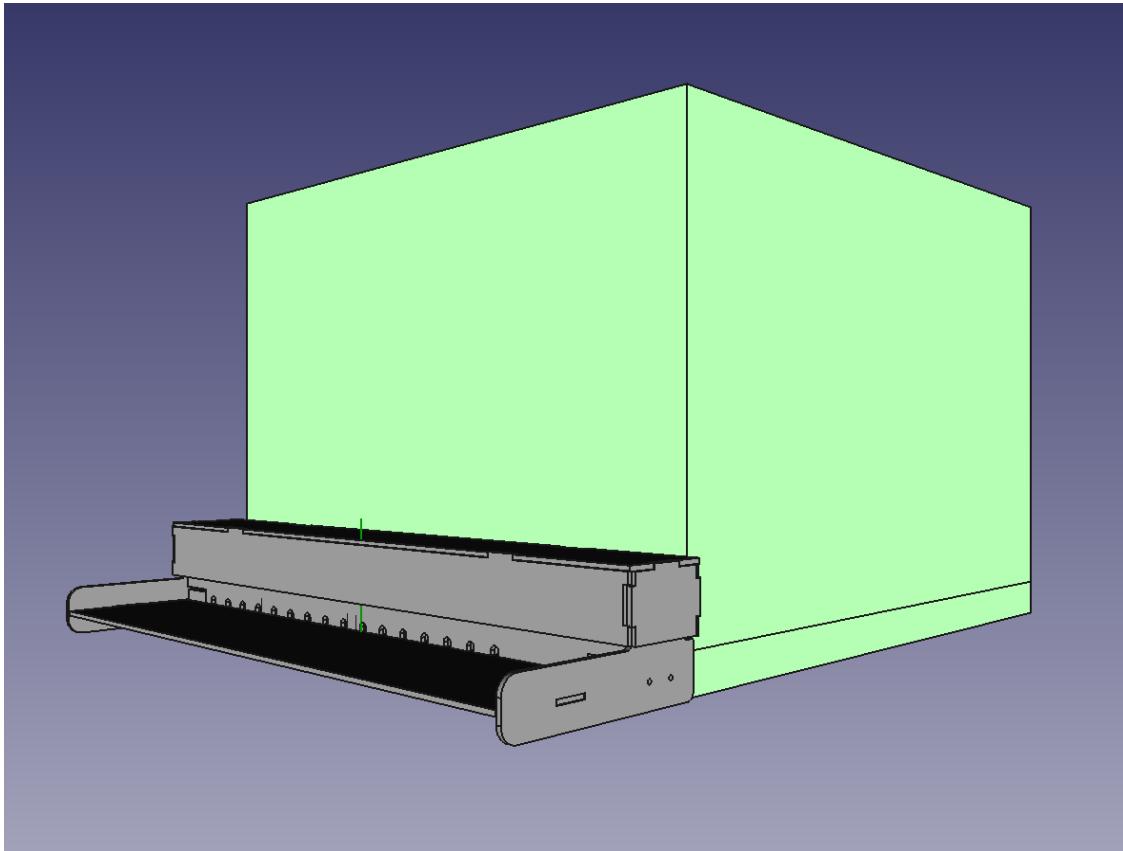
La carte SysBee est un matériel électronique qui est vulnérable aux phénomènes environnementaux tels que les intempéries ou la neige. Ainsi, il est important de la protéger de ces phénomènes qui pourraient compromettre son utilisation. Par conséquent, pour remédier à ce facteur nous avons eu l'idée de modéliser la maquette d'un support de la carte Sysbee sur le logiciel Freecad afin de lui permettre une protection optimale et durable. Cependant, pour produire cette maquette il faut respecter les dimensions de la carte afin de la rendre fonctionnelle et de faciliter son utilisation pour l'apiculteur.

1.1. Étude de l'enveloppe extérieure de la carte SysBee

1.1.1. Étude de la structure générale de l'enveloppe

Pour modéliser le support de la carte SysBee, nous avons utilisé Freecad qui est un logiciel de modélisation permettant de dessiner la structure que le veut et par la suite l'imprimer.

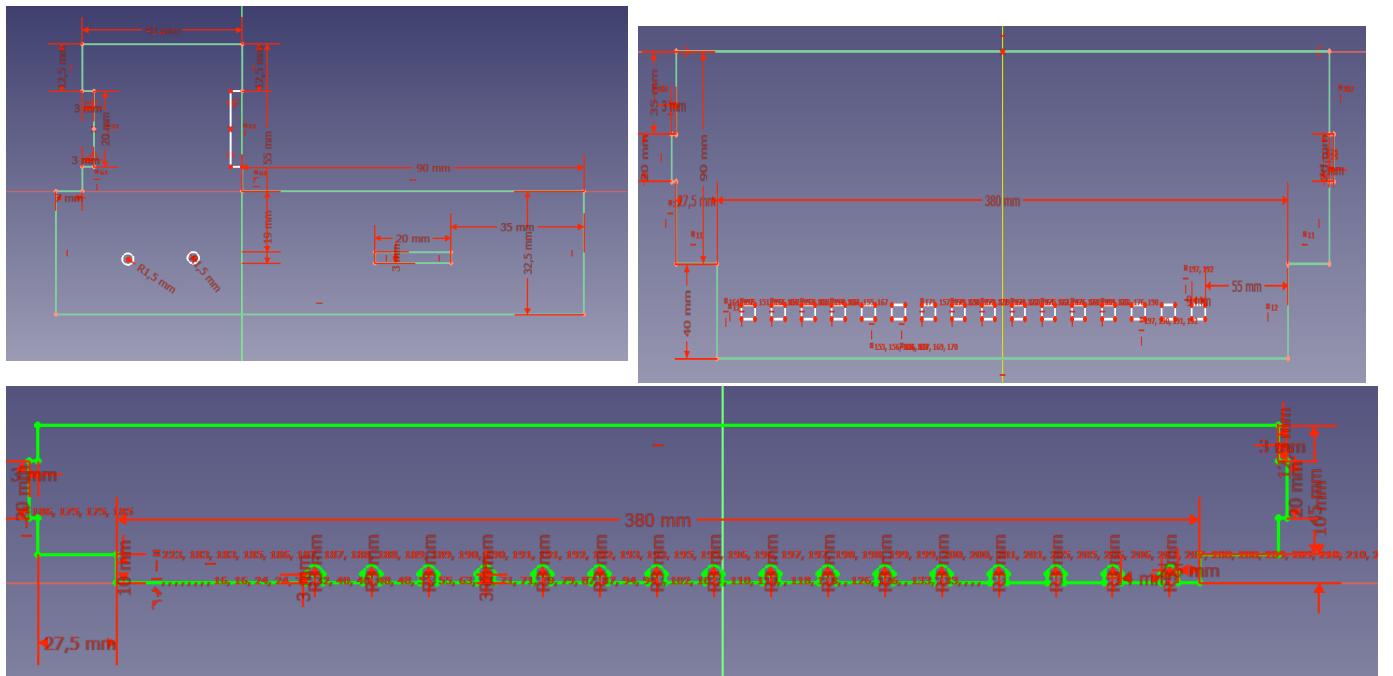




1.1.2. Adaptation à l'environnement

Tout d'abord, la carte SysBee est positionnée à l'intérieur d'un support (en gris sur la représentation ci-dessus), ce dernier est placé à l'entrée de la ruche (en vert). Le support de la carte SysBee est constitué de plusieurs éléments : les flancs, la planche d'envol, les peignes ainsi que la fermeture pour refermer le tout.

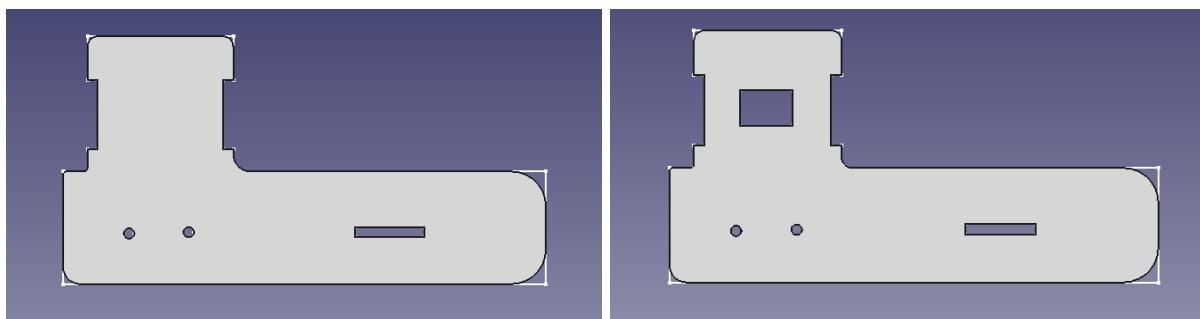
Chaque structure de la maquette est préalablement dessiné sur Freecad en 2D dans un premier temps :



Une fois que les dimensions sont respectées, nous pouvons extruder les composants afin de leur donner de la matière.

Voici la représentation de chaque éléments extrudés cités ci-dessous :

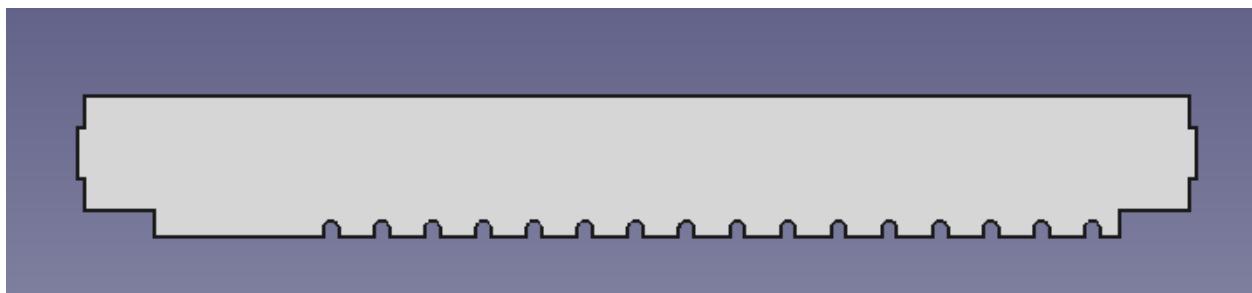
Les flancs, le second flanc possède une ouverture pour laisser passer le câble USB branché à la carte :



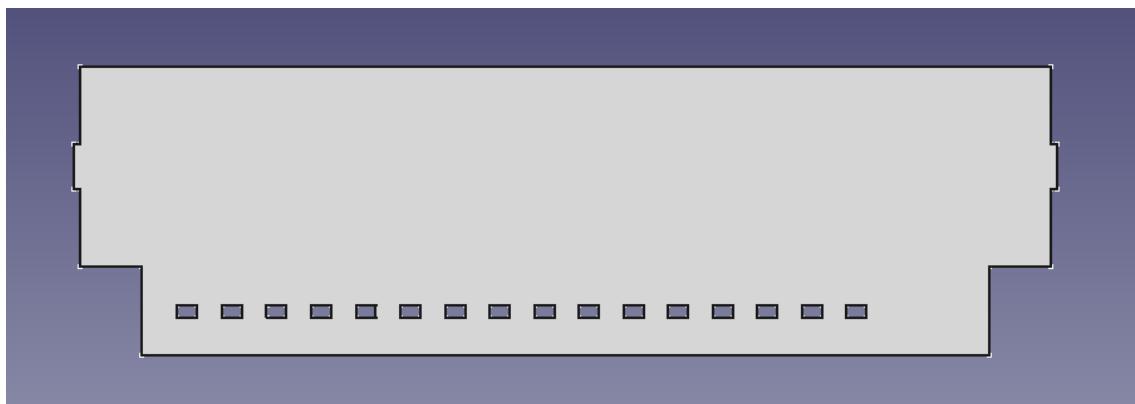
Les flancs servent de pilier pour la structure, en effet chaque flanc est positionné à chaque extrémité, la structure repose essentiellement sur ces flancs.

Le peigne possède 16 ouvertures permettant de faire entrer et sortir le maximum d'abeilles possible. L'objectif est que chaque abeilles puissent emprunter un seul unique chemin sous chaque capteur pour

que l'on puisse les compter. Les couloirs ont été réalisés pour être parfaitement alignés avec les capteurs pour rendre la détection des abeilles opérationnel. De plus, entre chaque couloir, des séparations ont été ajoutées afin que les abeilles ne puissent emprunter qu'une seule direction.

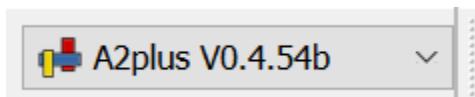


La planche d'envol est placée devant les entrées pour permettre aux abeilles d'atterrir ou de décoller plus facilement. Le rôle de la planche d'envol est de guider les abeilles afin qu'elles puissent pénétrer directement dans les couloirs de la structure et ainsi qu'elles puissent être détectées par le système de comptage.



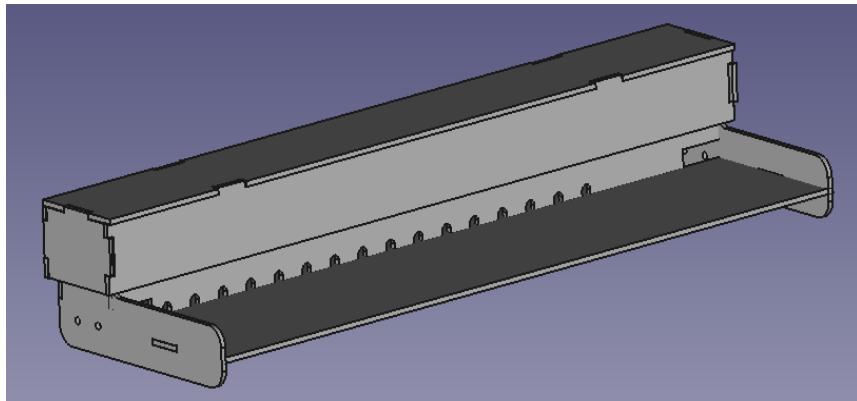
De plus, sur ces représentations chaque élément est constitué de tenons pour les consolider les uns entre eux.

Ensuite, lorsque tous les éléments ont été dessinés et extrudés nous pouvons passer à la partie assemblage qui consiste à vérifier si toutes les dimensions ont été respectées et par la même occasion de visualiser la représentation de la maquette.

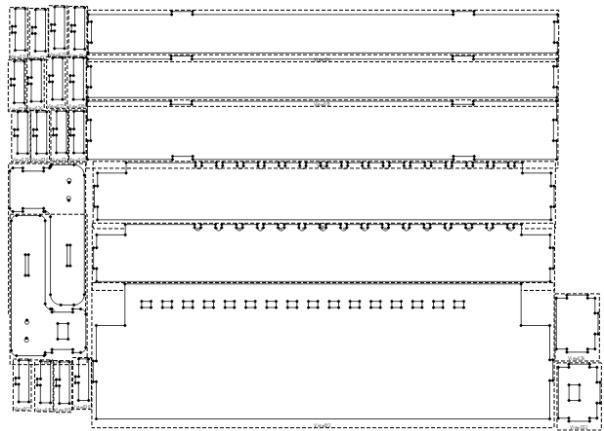


Le système a été conçu pour résister aux phénomènes météorologiques, il est ainsi protégé d'un couvercle le rendant résistant aux intempéries et est rendu étanche de part son assemblage à la colle à bois et aux tenons.

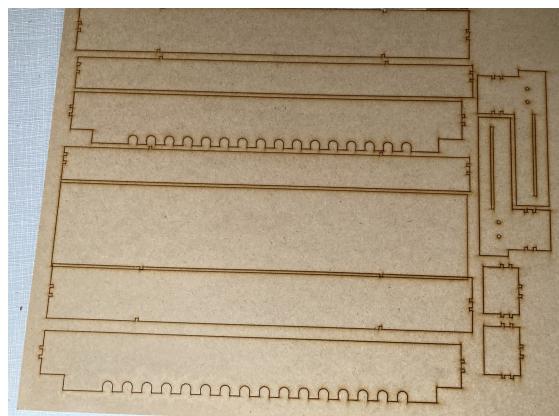
Assemblage final de la maquette :



Une fois l'assemblage terminé et que les dimensions correspondent parfaitement, nous l'avons mis en plan afin de le faire passer à la machine laser.



Mise en plan de tous les éléments



Après passage à la machine laser

La partie assemblage consiste à rassembler les éléments préalablement modéliser entre eux à l'aide de la colle en bois.



Photographie de la maquette finale du support de la carte SysBee.

Étudiant auteur de cette partie : Songchaiyavat SENGCHAN

1.2. Etude des capteurs optiques et des couloirs

Pour modéliser les couloirs de façon précise dans FreeCad, nous avons utilisé le modèle KiCad de la carte SyBee que nous avons pu transférer sur FreeCad à l'aide d'une Macro nommée "KiCadStepUp" qui permet le transfert entre ces deux logiciels.

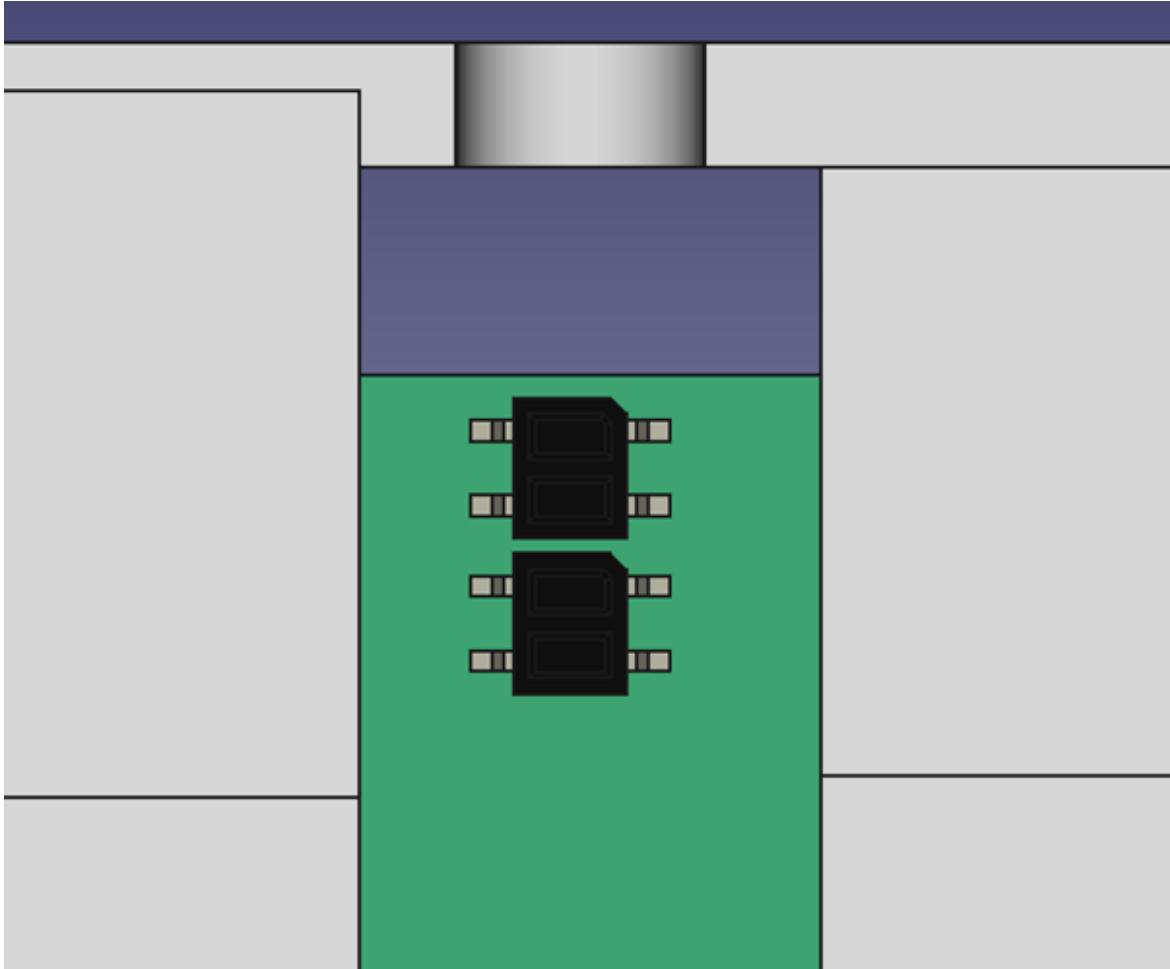


Figure 1. Porche optique

Les passages des abeilles sont détectés grâce à deux capteurs QRE1113 placés l'un derrière l'autre constituant ainsi un système de porche optique dans chaque couloir de la structure. La création de porche optique est nécessaire afin d'avoir un système de comptage fiable que l'on détaillera par la suite.

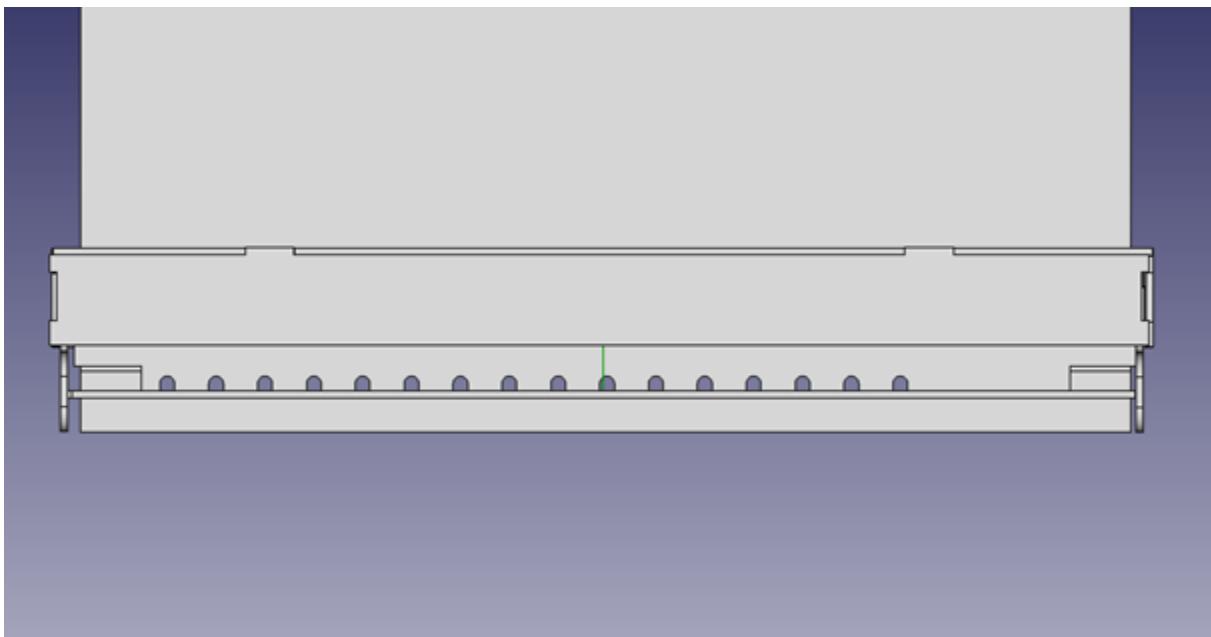


Figure 2. Couloirs vue de face

La structure possède 16 couloirs laissant entrer et sortir les abeilles qui sont situés devant la fente faite par les abeilles qui leur sert habituellement de passage pour circuler entre l'extérieur et la ruche. L'idée est qu'elles puissent emprunter un trajet bien défini sous chaque capteur pour que l'on puisse les compter.

Chaque couloir est conçu selon les dimensions d'une abeille, en hauteur et en largeur (hauteur : 6 mm et largeur 6 mm), seule la longueur des couloirs équivaut à la largeur de la carte SysBee qui est placée juste au-dessus des couloirs.

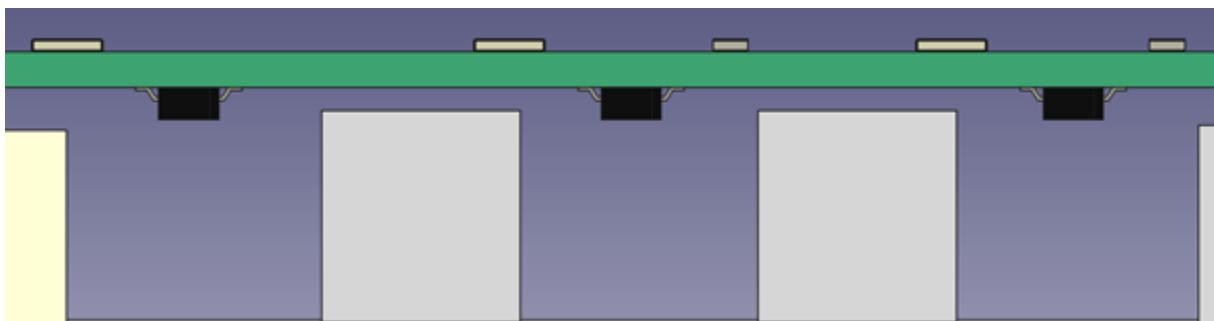


Figure 3. Alignement des capteurs

Les couloirs ont été conçus pour être parfaitement alignés avec les porches optiques pour rendre le système de comptage des abeilles opérationnel. Des séparations permettent de séparer chaque couloir les uns des autres afin que les abeilles ne faussent pas les mesures en empruntant différents couloirs. La carte Sysbee est située juste au dessus des séparations et est vissé sur ces derniers.

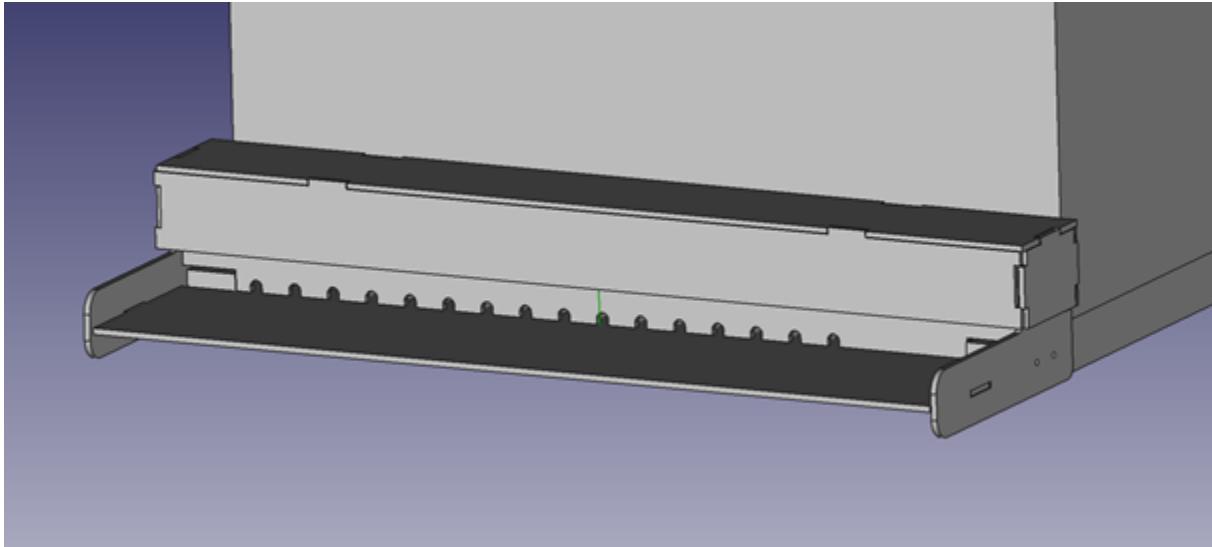


Figure 4. Planche d’envol

Une planche d’envol a été ajoutée au-devant de la structure pour permettre aux abeilles d’atterrir ou de décoller plus facilement. Le rôle de la plateforme est de guider les abeilles afin qu’elles pénètrent directement dans les couloirs de la structure et ainsi qu’elles puissent être facilement comptabilisées par le système de comptage.

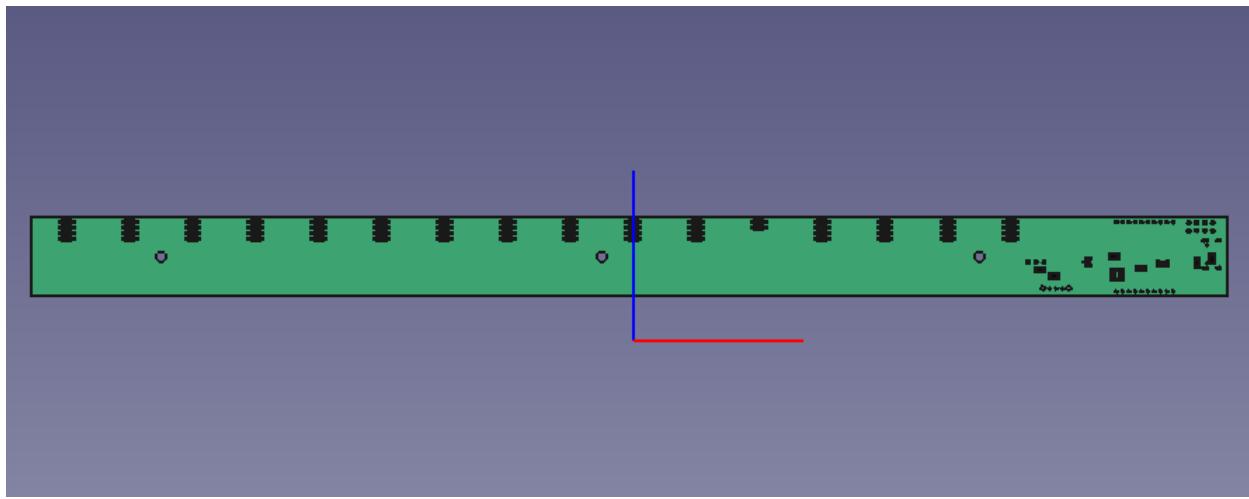


Figure 5. Carte SysBee vue de haut

La carte Sysbee possède 3 trous de fixation qui permettent de la fixer à la structure par l'intermédiaire des séparations avec des vis afin d'assurer sa stabilité pour ainsi éviter qu'elle ne se déplace à l'intérieur de la structure, ce qui pourrait fausser la détection des capteurs et donc fausser les résultats.

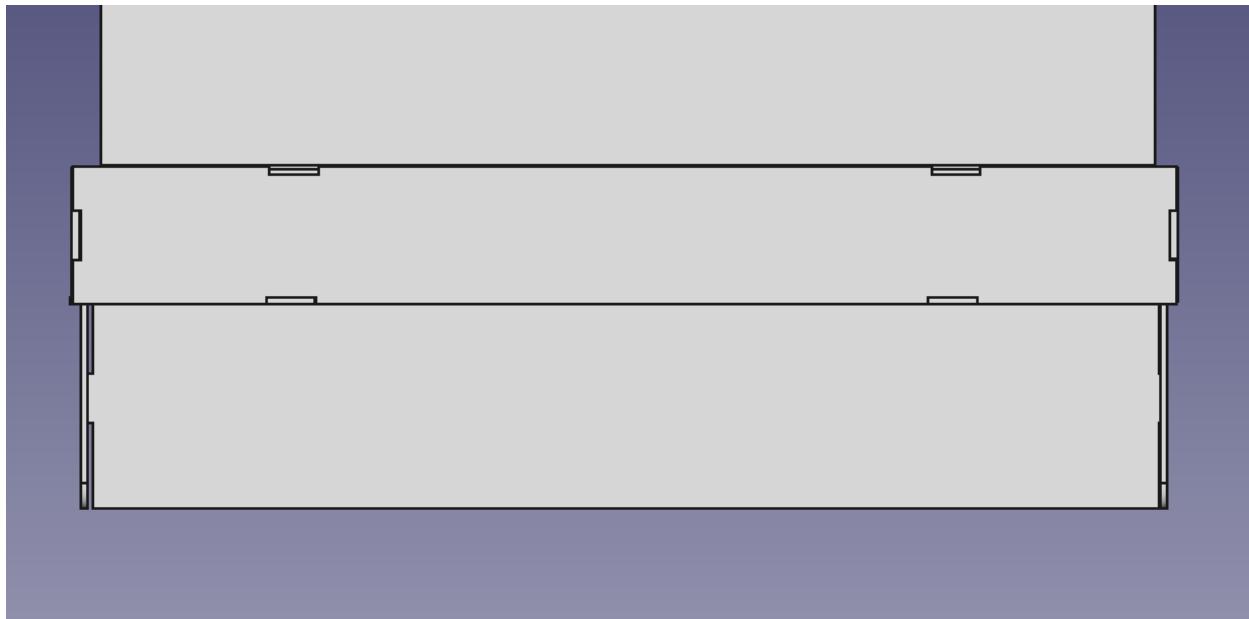


Figure 6. Couvercle de la structure

Le système a été conçu pour résister aux conditions extérieures, il est notamment doté de couvercles le rendant résistant aux intempéries et est rendu étanche de part son assemblage à la colle à bois et aux tendons. Cela est nécessaire, d'une part pour la protection de la carte SysBee et d'autre part pour empêcher les abeilles de circuler librement entre les couloirs et/ou d'éviter les porches optiques et ainsi fausser le système de comptage.

1.3. Conception du plateau test

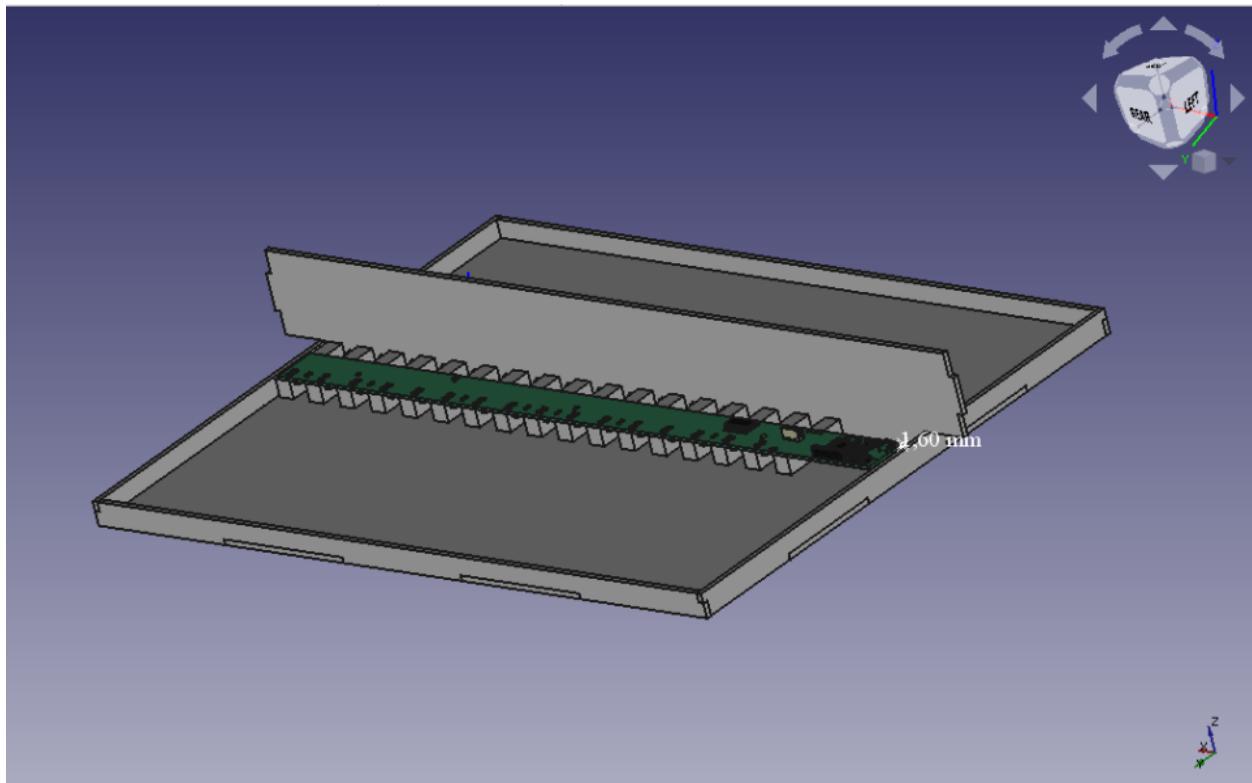


Figure 7. Modélisation du plateau test

L'impression d'un plateau test a été nécessaire pour effectuer les tests de bon fonctionnement de la carte SysBee. Des tests avec des billes jouant le rôle des abeilles circulant dans les couloirs permettent de réaliser des tests de bon fonctionnement de la carte SysBee. Le plateau test reprend le modèle de la structure générale mais positionné cette fois sur un plateau pivotable permettant de jouer sur le passage des billes inspiré d'un casse-tête labyrinthe à bille.

Étudiant auteur de cette partie : Ihsan DECIEUX

2 - Etude du matériel hardware de la carte SysBee

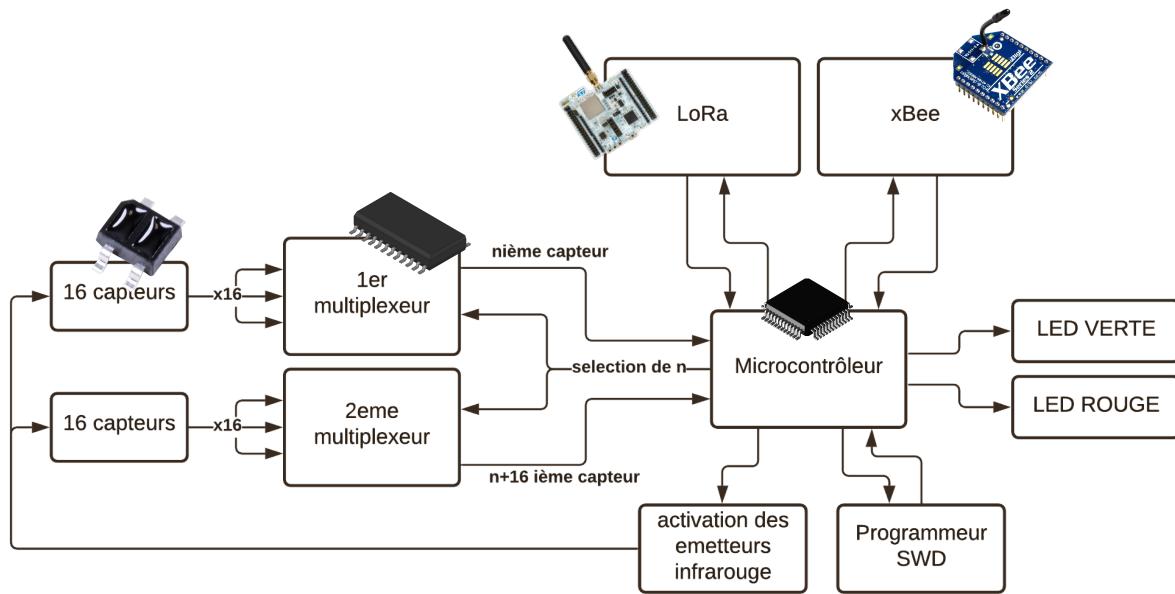
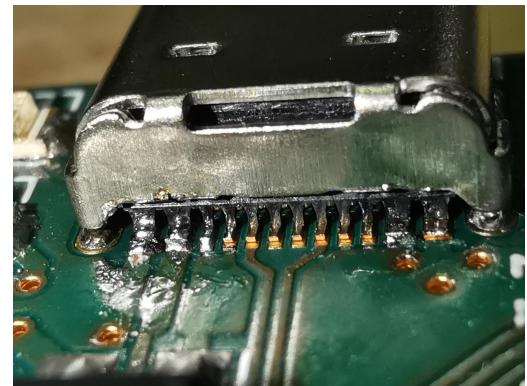


Schéma global de la carte

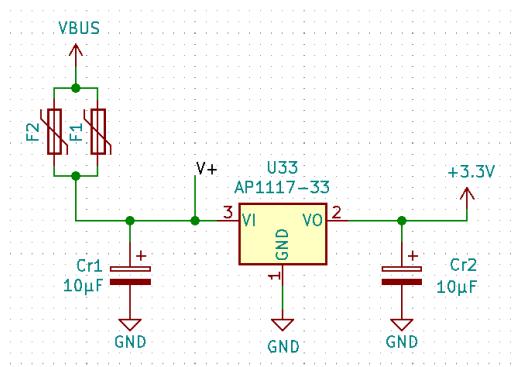
Le fonctionnement de la carte SysBee est plutôt simple. Elle mesure l'état des capteurs, en communiquant simultanément avec les modules radios que l'on détaillera plus tard.

2.1 : Alimentation

La carte mère SysBee dispose d'une prise USB-C qui lui fournit l'alimentation nécessaire. On peut donc l'alimenter avec une batterie portable solaire. Cela nous fournit une tension de 5V constants, sans risque de surtension, de sous-tension ni d'inversion. Comme la prise USB est arrivée tardivement, on a temporairement alimenté la carte avec des fils soudés sur les pads. (Sur la photo, les fils de données ne sont pas soudés parce que l'on n'avait pas encore accès au microscope)



On a tout de même inclus un fusible réarmable (polyfuse, deux en parallèle pour atteindre la valeur requise) afin de limiter le courant en cas de court-circuit. Ce sont des composants qui chauffent quand un courant les traverse, mais par rapport à une résistance classique, leur résistance augmente très vite passé une certaine température, ce qui coupe le courant jusqu'à ce qu'ils refroidissent totalement.



On abaisse les 5V à 3.3V pour le microcontrôleur et les multiplexeurs à l'aide d'un régulateur linéaire à faible chute. Aussi appelés LDO pour "Low DropOut", il permet de garantir une sortie de 3.3V à une tension d'entrée plus faible par rapport à régulateur classique.

Consommant un faible courant même en utilisant beaucoup les capteurs qui sont les plus énergivores, on peut s'attendre à une autonomie de 5 jours sur une batterie de 5000mAh.

Soit la capacité de la batterie **C = 5000mAh**, le courant moyen mesuré **i = 40mA** et le temps de fonctionnement t :

$$C = i \times t$$

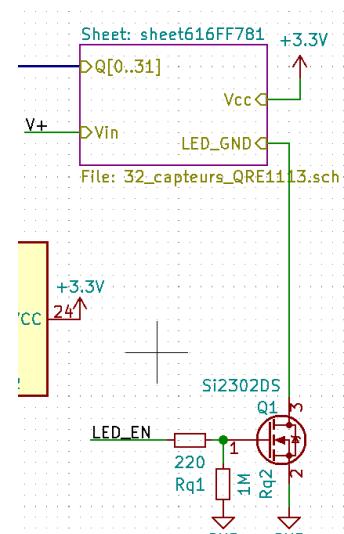
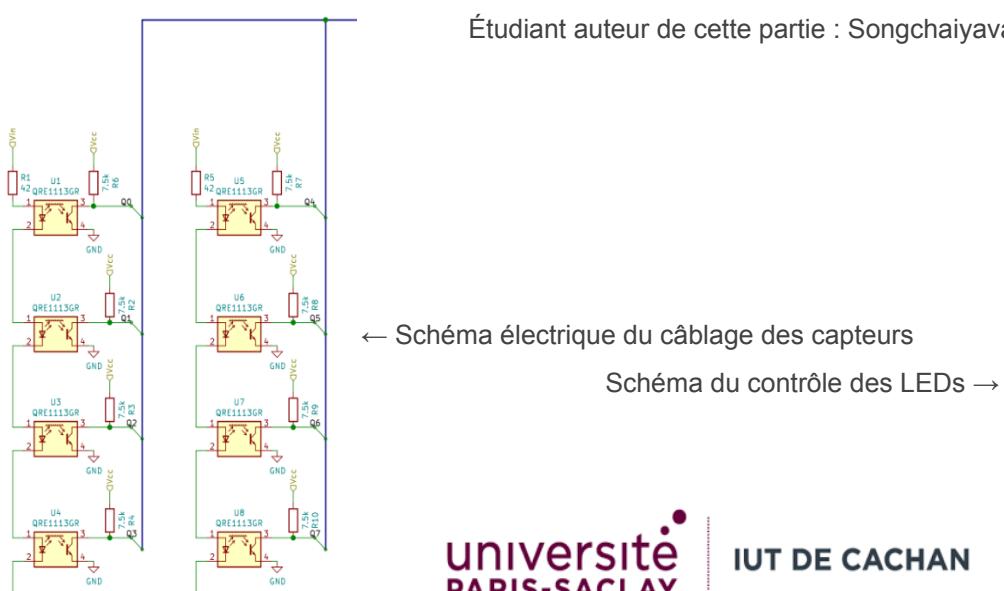
$$t = C / i = 5000 / 40 = 125 \text{ heures} = 5.2 \text{ jours}$$

Étudiant auteur de cette partie : Antoine SERRY

2.2 : Capteurs optiques :

Pour détecter le passage des abeilles, nous utilisons 32 capteurs optiques (QRE1113). Ils sont composés d'un émetteur (une diode), et d'un un récepteur (un phototransistor). Cette diode à une tension directe de 1.2V au courant utilisé. C'est-à-dire qu'on peut en mettre 4 en série pour se rapprocher des 5V de l'alimentation. On a donc dû mettre en parallèle 8 séries de 4 émetteurs. Afin d'économiser de l'énergie, on peut désactiver les LEDs quand on n'a pas besoin de détecter un essaimage, ou qu'on le détecte lentement (mode échantillonnage éparse que l'on verra plus tard). On fait cela à l'aide d'un MOSFET canal N afin de contrôler le courant avec une broche du microcontrôleur. On inclut une résistance de grille afin de limiter le courant dû à la capacité de la grille, et ainsi protéger le GPIO. Une résistance de tirage vers la masse permet d'éviter que le transistor soit en région ohmique si une erreur de programmation rend le GPIO flottant, ce qui pourrait l'endommager.

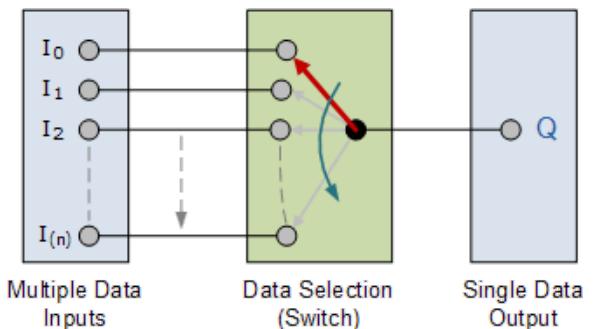
Étudiant auteur de cette partie : Songchaiyavat SENGCHAN & Antoine SERRY



Le phototransistor laisse passer un courant plus fort lorsqu'il reçoit plus des infra-rouges réfléchis. En plaçant une résistance de charge entre Vcc (3.3V) et son collecteur on peut convertir le courant en tension lisible par un circuit logique. Afin d'atteindre ses niveaux logiques, on a ajusté la résistance de charge pour avoir une détection à distance du dos de l'abeille, sans détecter le sol du couloir. La sortie vaut Vcc quand rien ne réfléchit la lumière, et 0V quand une abeille se trouve devant. Après une tentative infructueuse de calculer la résistance à partir de la datasheet du capteur, on a fait un produit en croix avec la résistance actuelle, la distance de détection obtenue, et désirée. Cette fois, on a eu la bonne résistance du premier coup.

2.3 : Multiplexage :

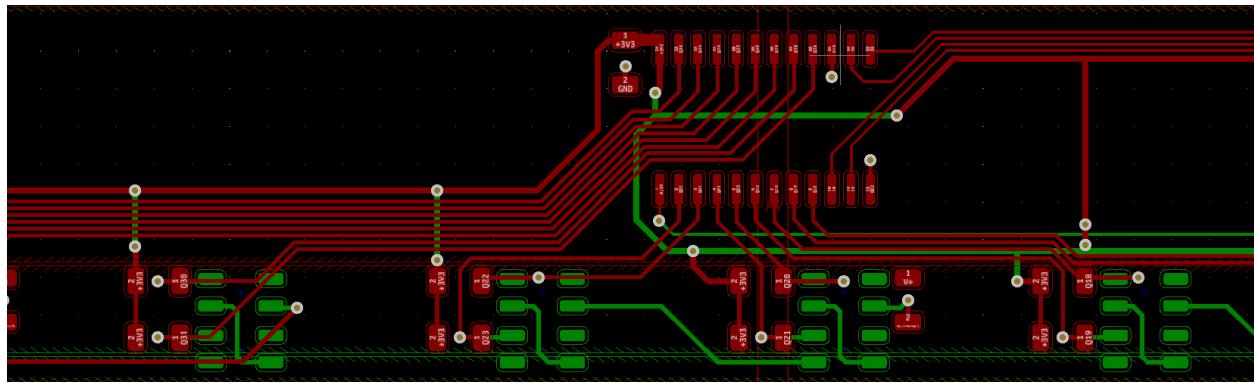
Comment récupérer les 32 bits venant des capteurs avec un microcontrôleur au nombre de pins limités ? En utilisant des multiplexeurs. Les plus adaptés que l'on ait trouvé ont 16 entrées. On en prend donc deux. Ils relient l'entrée de notre choix à leur unique sortie. Tout ce qu'on a à faire c'est choisir l'entrée avec 4 bits de sélection ($\log_2(16) = 4$). On a mis les 4 mêmes bits de sélection sur les deux multiplexeurs afin de simplifier la programmation et le routage. Le logiciel se chargera donc de parcourir les entrées et d'enregistrer la valeur correspondante.



Physiquement, ils sont placés au plus près des capteurs dont ils s'occupent. On a veillé à ce que les capteurs du même couloir se suivent, et toujours dans le même ordre sur les entrées du MUX pour faciliter la programmation. Surtout, ils sont sur la face supérieure du circuit pour ne pas interférer avec le montage, une fois mis à plat sur les porches optiques. Ci-dessous la partie du routage en question et une vue en détail.



Portion du routage où se trouvent les MUX et les capteurs



Les plans (zones remplies) ont été rendus invisibles pour faciliter la compréhension du routage.

2.4 : Microcontrôleur et composants auxiliaires

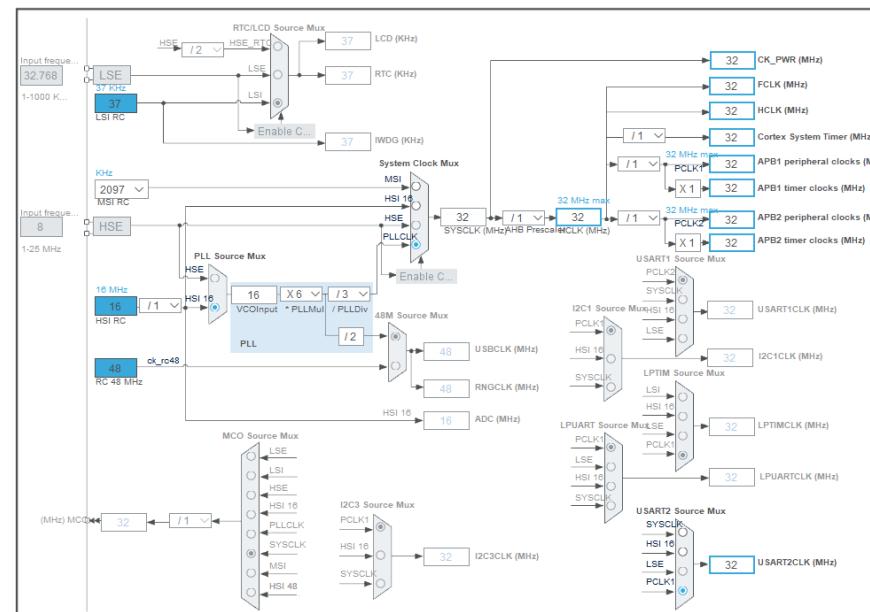
Le microcontrôleur que l'on utilise est le **STM32L073rz**. Il fait partie de la gamme "Basse consommation" du fabricant STMicroelectronics mais ne manque pas de ressources pour autant. En effet, il peut être cadencé jusqu'à 32 MHz et dispose d'une mémoire amplement suffisante pour nos besoins.

Il dispose de broches utilisables pour le programmer. On a ajouté un connecteur MiniConnect pour le brancher au programmeur SWD (Serial Wire Debug, un variant de JTAG) une fois brasé sur le PCB, qui établit la communication avec un ordinateur.

Une fois bien branché peut commencer à le paramétrer dans STM32CubeIDE.

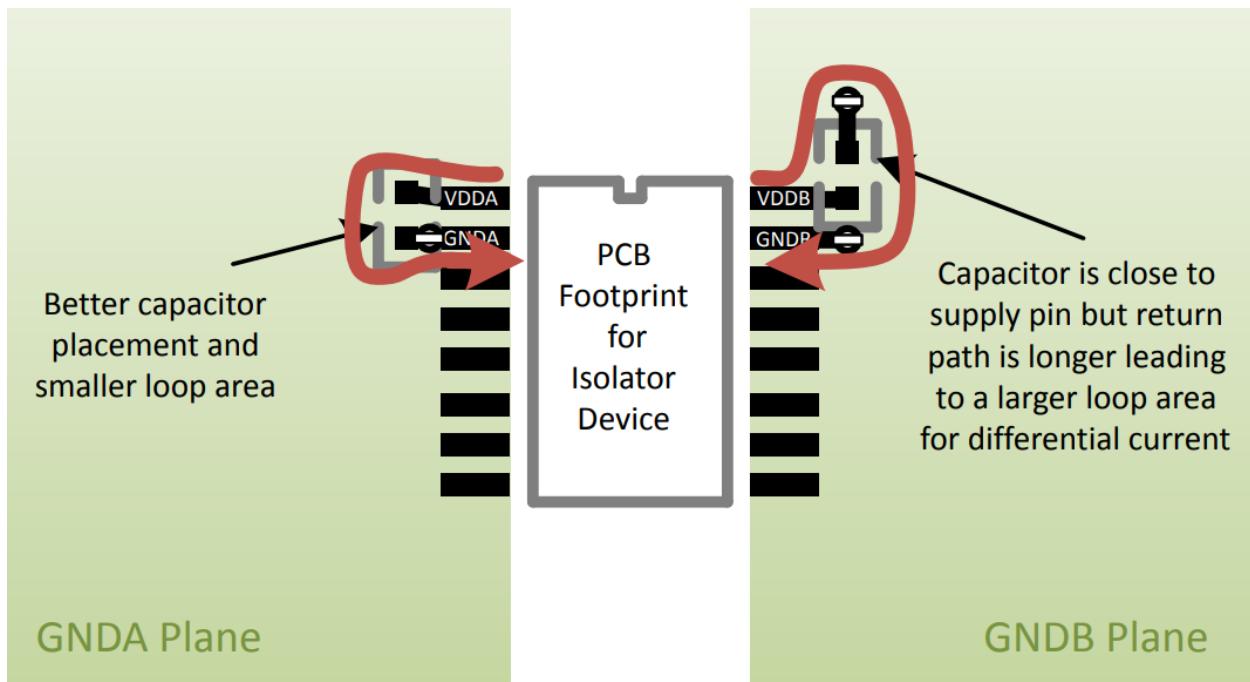
Ce microcontrôleur dispose d'une horloge interne paramétrable. On a envisagé d'ajouter un oscillateur externe pour plus de précision, mais notre application n'en demande pas autant (même les interfaces UART qu'on utilise à seulement 9600 b/s).

On a veillé à bien avoir 48 MHz sur l'horloge USB pour permettre son fonctionnement



Panneau de configuration de l'horloge

Le microcontrôleur a plusieurs entrées d'alimentation. On doit veiller à toutes les connecter et à mettre un condensateur de découplage au plus proche de chacune d'elles. Il faut veiller à minimiser la surface de la boucle que prend le courant, et sa longueur.



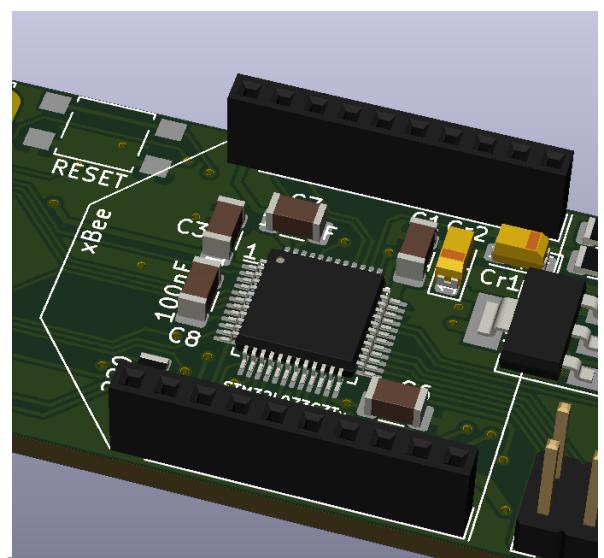
"Minimize Loop Area with Optimum Bypass Capacitor Placement" - an1131

Afin de comprendre facilement l'état du système, on a ajouté trois leds. Une rouge qui s'allume aussitôt que le circuit est alimenté. Les deux autres, une rouge et une verte, sont contrôlées par le microcontrôleur et suivent des séquences de clignotement représentant chaque état.

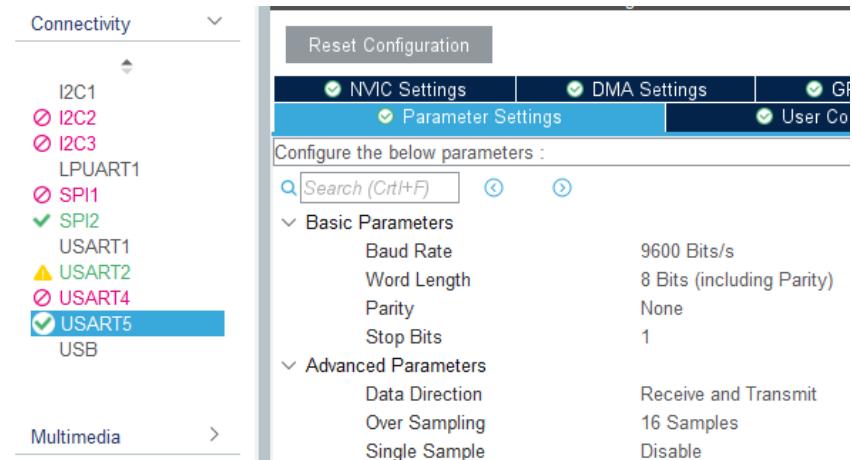
2.5 Périphériques de communication :

Comme évoqué précédemment, le circuit dispose de plusieurs moyens de communiquer avec d'autres appareils. Le port SWD avec un connecteur MiniConnect, mais aussi SPI pour le module LoRa, et enfin UART avec le module xBee. On a disposé des connecteurs femelles de part et d'autre du circuit pour accueillir le xBee, sans oublier un détrompeur dessiné sur le SilkScreen.

Emplacement xBee et détrompeur



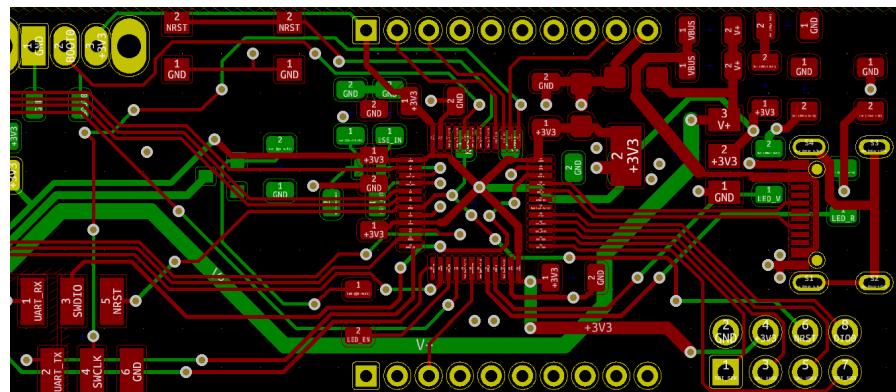
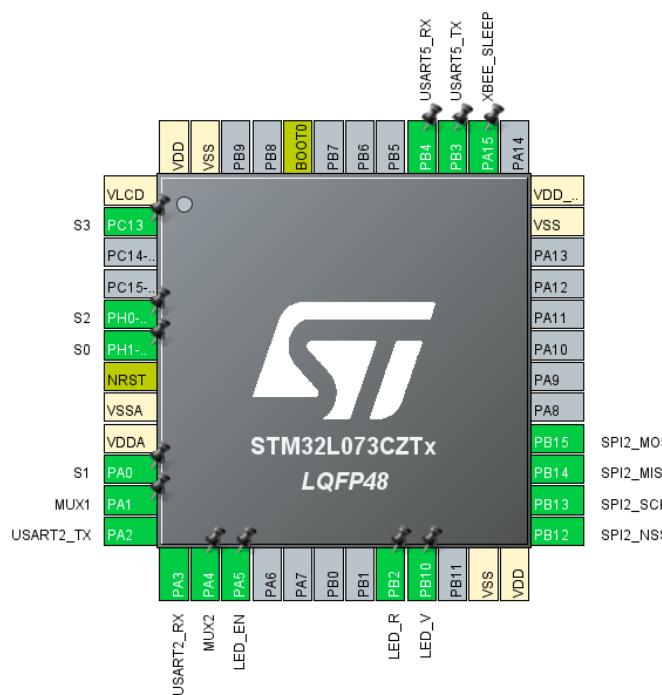
Ces périphériques sont paramétrables directement dans cubeMX (cf l'interface utilisée).



Interface cubeMX, paramétrage UART

2.6 : Fabrication et test :

Afin de router la carte efficacement, on a choisi les entrées et les sorties avec attention. On place sur le microcontrôleur les GPIO pas nécessairement au plus proche, mais surtout dans un ordre qui permet d'éviter les croisements. Ci-dessous l'outil utilisé et le résultat.



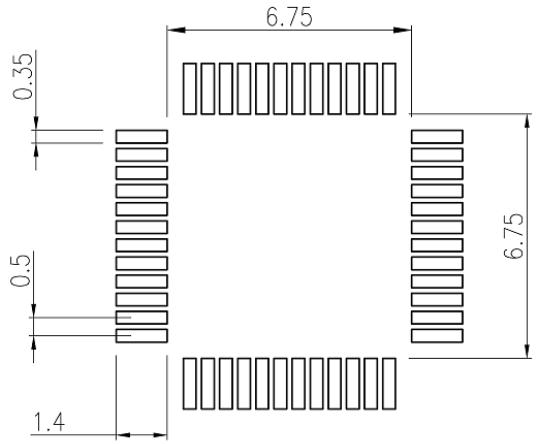
Outil de configuration des pins sur CubeMX

/

Routage une fois terminé

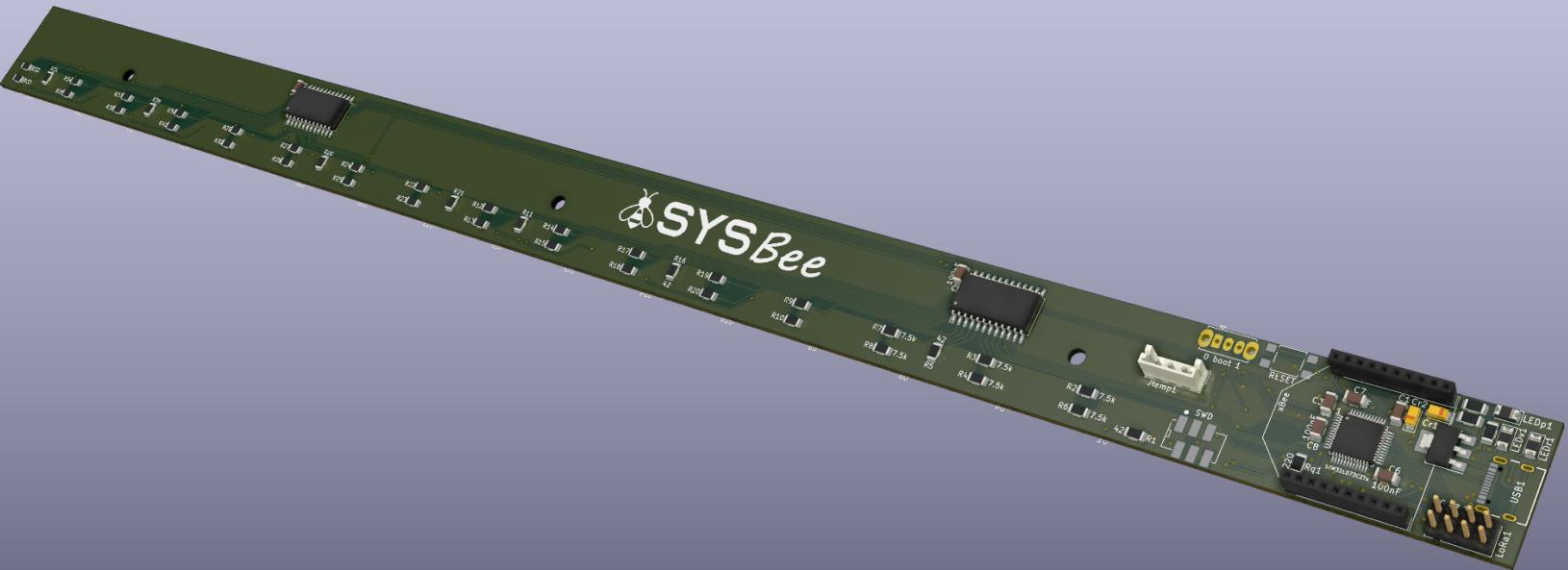
Une fois le routage terminé, on a créé les fichiers nécessaires à envoyer au fabricant. Quelques semaines plus tard, on a reçu le PBC et il ne restait plus qu'à braser les composants. On a commencé par le plus délicat : le microcontrôleur afin d'avoir de ne pas être gêné par les autres composants. La loupe binoculaire nous aura été d'une grande utilité, non seulement pour pouvoir viser précisément les pattes, mais aussi pour inspecter le circuit qui, on le verra, nous a causé quelques problèmes.

Emprunte LQFP-48 du microcontrôleur (unité mm) →



Quand tous les composants ont été soudés, on a pu tester la carte. En l'alimentant avec des fils à défaut d'avoir la prise USB comme évoqué précédemment. Nous l'avons branchée à une alimentation de laboratoire à l'aide de fiches banane. On limite le courant en cas de dysfonctionnement (foreshadowing). Cette limite a malheureusement été atteinte dès l'allumage. Nous avons donc procédé à une inspection complète du circuit au microscope, infructueuse. En retirant le régulateur on en a déduit que le court-circuit se trouvait sur l'alimentation 3.3V. Malgré ça, aucun pont de soudure, ni de composant manquant. Le microcontrôleur a donc été désoudé à l'infrarouge comme il est le plus susceptible d'être mal soudé. C'est là que l'on remarque que le cuivre d'une patte, sous le vernis épargne, s'étendait jusqu'à la masse. On l'a donc délicatement coupé avec un scalpel sans détériorer le reste. Un nouveau microcontrôleur en place, et le problème était résolu. Le premier test que l'on a effectué pour s'assurer du fonctionnement était de faire clignoter les voyants LED.

Étudiant auteur de cette partie : Antoine SERRY



Rendu numérique du PCB

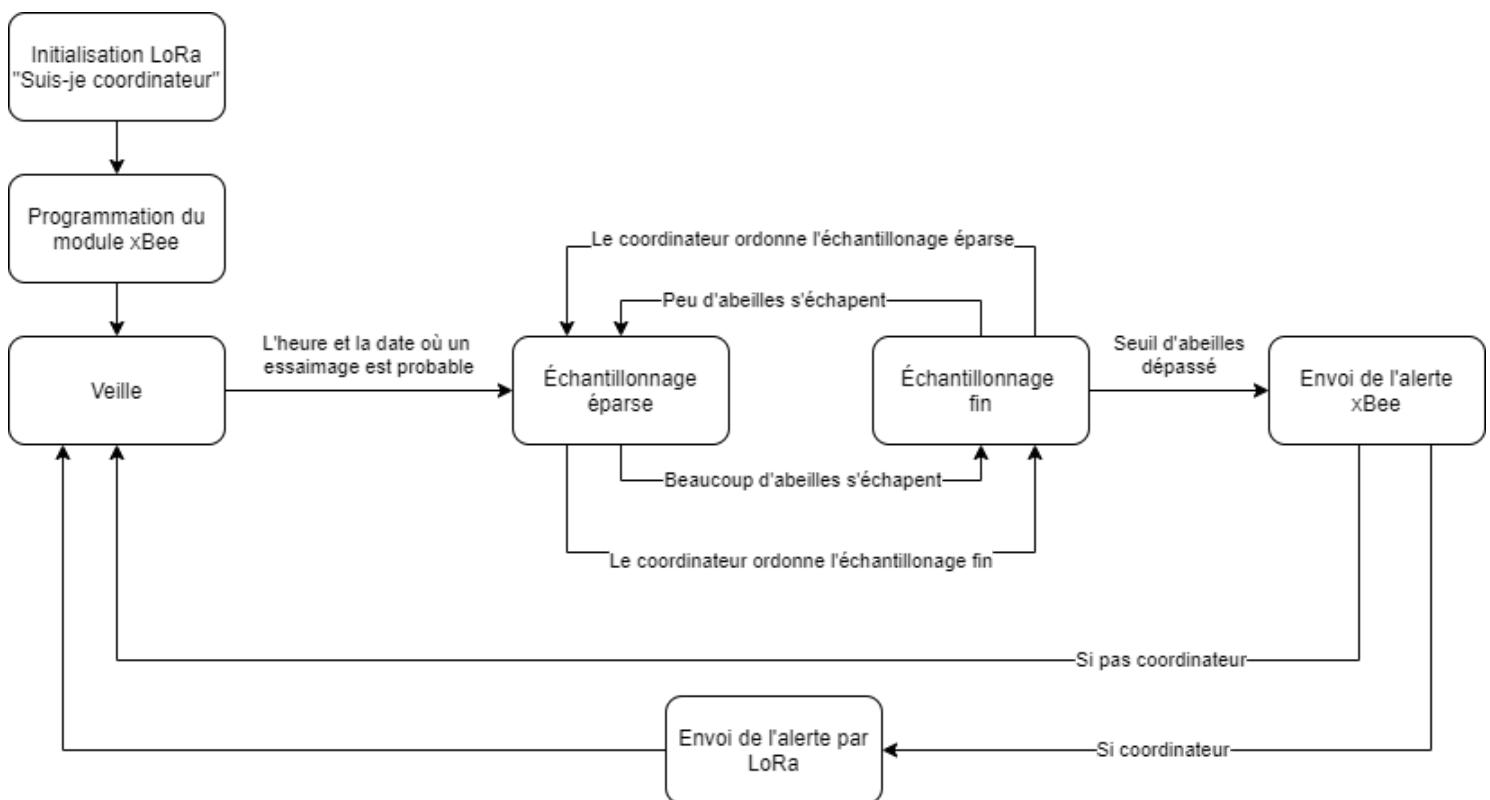
3 - Programmation, fonctionnement du logiciel :

Le principal intérêt de notre méthode est que l'on peut faire un réseau de cartes SysBee totalement identiques. C'est-à-dire qu'elles ont le même programme, et le même matériel. N'importe laquelle peut endosser les rôles que l'on décrira tout de suite après. Cela facilite non seulement la production si on voulait en faire d'autres, mais aussi l'utilisation. Tout est automatique, et l'apiculteur a très peu d'action à réaliser pour faire fonctionner le réseau.

Ces rôles sont "coordinateur", et "esclave". Le "coordinateur" est la ruche qui communique avec l'apiculteur par LoRa, elle peut envoyer des instructions aux autres ruches "esclaves". Elle sait si elle est coordinateur simplement en la branchant au module LoRa. Les autres sont "esclaves" par défaut. Grâce au LoRa, le coordinateur peut connaître la date et l'heure. Elle ordonne donc aux autres de passer en mode veille quand un essaimage est impossible pour économiser de l'énergie. C'est aussi elle qui va recueillir les alertes d'essaimage des esclaves, et les transmettre à l'apiculteur.

3.1 : Automate global :

Le logiciel de SysBee peut se décomposer en plusieurs états qui ont chacun leur propre utilité. On a un automate global représenté ci-dessous, dans lequel chacun des états est composé d'autres automates.



3.2 : Échantillonnage et comptage :

Pour compter efficacement les abeilles, on doit pouvoir savoir dans quel sens elles se déplacent, mais aussi rejeter les informations si une abeille reste sous un porche.

Pour cela il faut dans un premier temps reconstituer les signaux d'un porche c'est-à-dire sélectionner le premier capteur avec le multiplexeur, enregistrer la valeur, passer au deuxième, et on obtient deux bits représentant les deux capteurs du porche.

Lorsqu'une abeille traverse le couloir, elle déclenche un capteur, puis l'autre, libère le premier, puis le deuxième. L'ordre dépend uniquement de la direction de l'abeille. Ce signal est très similaire à celui d'un encodeur à quadrature. Si ressemblant qu'on peut utiliser le même automate pour compter les abeilles.

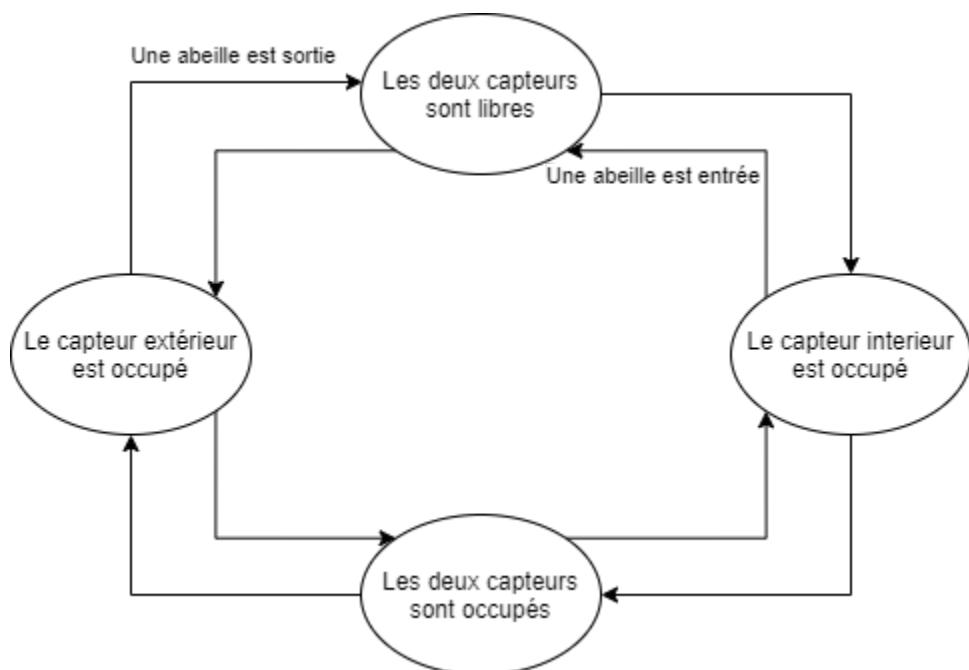


Diagramme d'état du compteur d'abeilles

Chaque porche possède une variable stockant les valeurs précédentes afin de savoir les transitions qu'il peut compter, ou négliger (les transitions possibles sont représentées par des flèches sur le diagramme). Lorsqu'on fait un tour complet du diagramme, on incrémente ou diminue le nombre d'abeilles. On fait donc une boucle "**for**" qui, pour chaque porche, va mesurer les deux capteurs, mettre leurs valeurs dans l'automate, et en déterminer l'état suivant. (Programme à la page suivante)

L'explication ci-dessus s'applique aux deux fonctions, échantillonnage éparse (sparse polling) et échantillonnage fin (fine polling). La différence est que dans fine polling, on laisse le programme s'exécuter aussi vite que possible pour ne rien rater. A l'inverse, le sparse polling va faire une mesure, éteindre les capteurs pour économiser de l'énergie, puis attendre avant de recommencer. Ça veut dire que ce mode est 90% moins énergivore que fine polling.

```

void scan_sensors() {

    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, 1); //turn on the diodes

    for(uint8_t i=0; i<8; i++) for(uint8_t j=0; j<2; j++) {
        uint8_t sensor_pair_value;

        //select inside sensor of the current pair
        select_mux(sensor_pair[8*j+i][0]);

        //get inside sensor value
        sensor_pair_value = (j) ? HAL_GPIO_ReadPin(MUX1) : HAL_GPIO_ReadPin(MUX2);

        //select outside sensor of the current pair
        select_mux(sensor_pair[8*j+i][1]);

        //get outside sensor value
        sensor_pair_value += (j) ? HAL_GPIO_ReadPin(MUX1)<<1 : HAL_GPIO_ReadPin(MUX2)<<1;

        switch(sensors_state[8*j+i]) { //state machine
            case A:v
                if(sensor_pair_value == 0b01){ //increment or decrement bee count
                    sensors_state[8*j+i] = B;
                    bee_count--;
                }
                else if(sensor_pair_value == 0b10){
                    sensors_state[8*j+i] = D;
                    bee_count++;
                }
            break;

            case B:
                if(sensor_pair_value == 0b00) sensors_state[8*j+i] = C;
                else if (sensor_pair_value == 0b11) sensors_state[8*j+i] = A;
            break;

            case C:
                if(sensor_pair_value == 0b10) sensors_state[8*j+i] = D;
                else if (sensor_pair_value == 0b01) sensors_state[8*j+i] = B;
            break;

            case D:
                if(sensor_pair_value == 0b11) sensors_state[8*j+i] = A;
                else if (sensor_pair_value == 0b00) sensors_state[8*j+i] = C;
            break;
        }
    }
    //turn off the diodes if in sparse polling state
    if(master_state == sparse_polling) HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, 0);
}

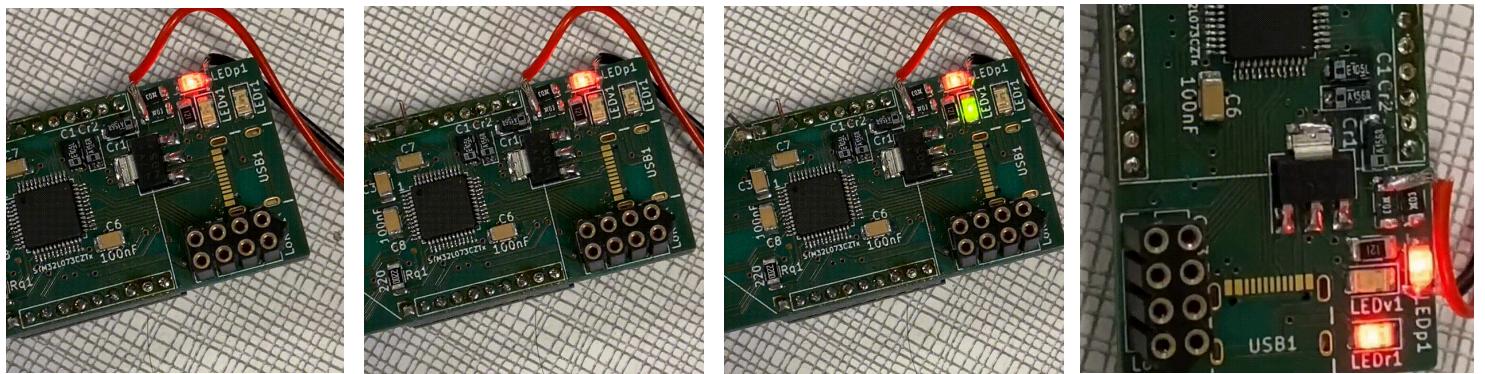
```

Une fois que l'on sait quand une abeille sort ou rentre, on utilise un ticker pour faire une dérivée, c'est à dire qu'on ne compte plus seulement les abeilles, mais le nombre d'abeilles qui s'échappent en un temps donné. Si cette dérivée dépasse un seuil réglable (environ 5000 abeilles / minute) alors on envoie une alerte essaimage.

Étudiant auteur de cette partie : Antoine SERRY

3.3 : Signaux lumineux :

On a étudié le programme afin de simplifier l'utilisation au maximum, cependant il peut quand même être utile de savoir ce que fait le système. Pour cela on peut imaginer un log envoyé au téléphone de l'apiculteur etc... Mais le plus simple reste encore des voyants lumineux. Ils doivent émettre des signaux visuels clairs et sans ambiguïté. Voilà à quoi ils ressemblent (ça marchera moins bien sur la version papier).



Initialisation xBee

Fonctionnement normal

Alerte essaimage

Défaut matériel

Initialisation : Vert clignote rapidement;

Fonctionnement normal : vert fait deux bips comme un battement cardiaque;

Alerte essaimage : Vert et Rouge s'alternent rapidement;

Défaut matériel : Rouge clignote rapidement

Les signaux doivent s'exécuter en parallèle du fonctionnement normal, c'est pourquoi il ne peuvent pas être faits avec des fonctions bloquantes comme un "delay". Ils doivent aussi être faciles à modifier et à ajouter. C'est pourquoi on a créé un tableau à 3 dimensions. La première détermine quel signal on veut. La deuxième détermine l'étape du signal. La troisième détermine la led concernée par la valeur stockée.

```
uint8_t signals[5][20][2] = {
{
    // {r, v} --> ok - deux bips rapides puis une pause
    {0, 1}, //bip
    {0, 0},
    {0, 1}, //bip
    {0, 0},
    {0, 0}, // pause...
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0},
    {0, 0}
},
```

```
// etcetera
```

Pour les lire, on utilise un ticker avec une période de 50ms. On l'obtient avec CubeMX (cf la capture d'écran ci-dessous), en allumant un timer, on calcule son prescaler et sa période d'overflow à partir de la période voulue, et sa fréquence d'incrément. Voici la formule qui relie ces valeurs

$$T_{out} = \frac{T_{overflow} * Prescaler}{Fin}$$

Paramétrage du compteur du ticker →

▼ Counter Settings

Prescaler (PSC - 16 bits value)	65535
Counter Mode	Up
Counter Period (AutoReload R...)	4883
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

Le ticker parcours les valeurs du signal demandé et les applique aux leds à une vitesse comme montré dans le programme suivant :

```
void update_signal() { // s'exécute toutes les 50ms
    static uint8_t i = 0;
    HAL_GPIO_WritePin(GPIOB, LED_R, signals[signal_state][i][0]);
    HAL_GPIO_WritePin(GPIOB, LED_V, signals[signal_state][i][1]);
    i = (i+1)%20;
}
```

Comme on l'a vu précédemment, le meilleur moyen d'économiser de l'énergie est de couper les capteurs la plupart du temps. Mais on peut aussi paramétrier le module xBee pour qu'il passe en veille quand il n'est pas utilisé. Le compromis, c'est que sans ça, les ruches peuvent se relayer une information avant d'atteindre le coordinateur si il était hors de portée de l'émetteur. Ce n'est plus possible si ces relais s'éteignent périodiquement à moins de complexifier le programme.

Étudiant auteur de cette partie : Antoine SERRY

4 - Communications.

4.1 : Programmation du xBee par UART :

Pour savoir quelle ruche a un essaimage, on doit donner un nom unique à chacune d'elles. Ce nom est stocké sur le module xBee. Comme on voulait que le programme soit identique sur toutes les cartes SysBee, il fallait que la carte puisse reprogrammer son xBee. Pour cela, il y a une méthode bien précise. Il faut envoyer trois octets “+++” par UART pour passer en mode configuration. A chaque fois qu'on envoie une commande, on doit attendre que le xBee réponde “OK” ou réessayer si on a pas de réponse après un moment. Puis on peut envoyer “AT” suivi du code correspondant au paramètre, et la valeur qu'on veut y mettre. Ces codes de paramètres peuvent être trouvés dans la documentation du xBee. Enfin on sauvegarde en envoyant “ATWR”. Imaginons qu'on veuille changer l'identifiant du réseau pour “1234” et le nom du xBee pour “nom”. La discussion serait celle-ci :

```
→ +++  
← OK  
→ ATRE ( pour réinitialiser le xBee, on ne connaît pas les paramètres qui ont pu être dessus avant )  
← OK  
→ ATID 1111  
← OK  
→ ATNI nom  
← OK  
→ ATWR  
← OK
```

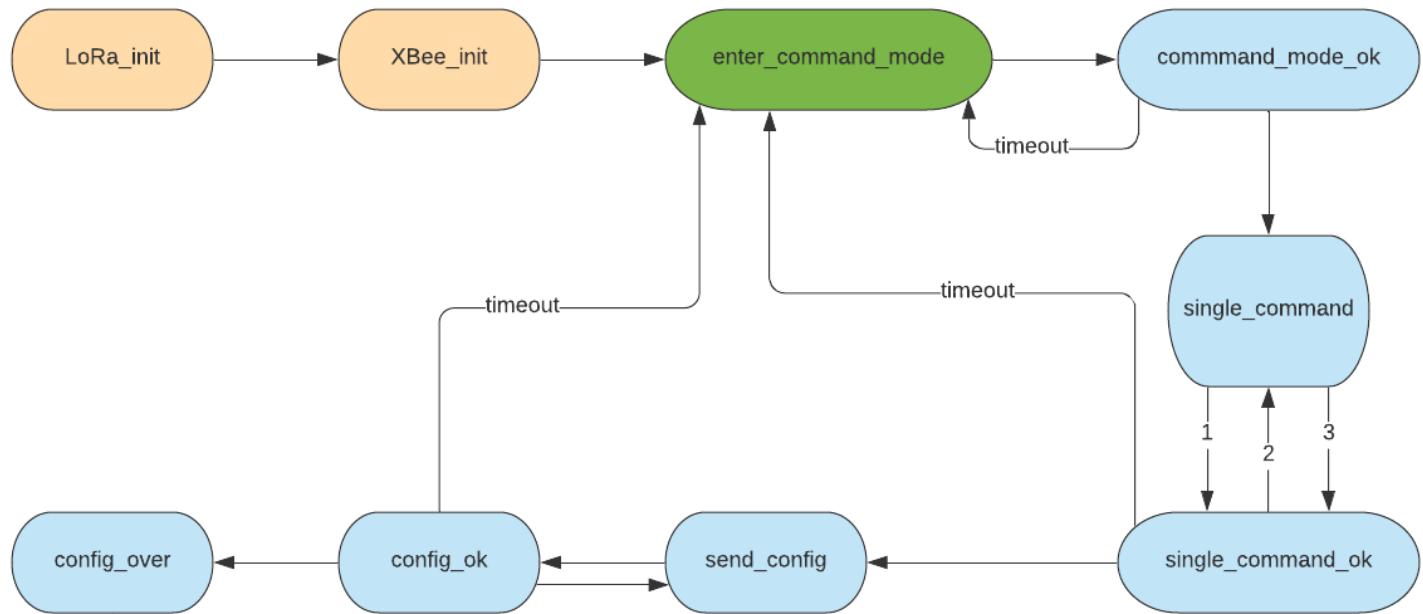
Si on ne reçoit pas “OK” au bout d'un certain temps, on reprend à “+++”. Si on n'a toujours pas de réponse au bout de 5 tentatives, c'est probablement à cause d'un défaut matériel donc on entre dans le mode “fault”.

On a un tableau afin de stocker tous les codes de paramètres, et leurs valeurs correspondantes. En voici la déclaration.

```
uint8_t config[20][2][10] = {
    {"ID", "1111"}, // network id
    {"NI", ""}, // node id
    {"CE", "0"}, // is coordinator
    {"ID", "1111"}, // network id
    {"AP", "1"}, // API enable
    {"SP", "20"}, // cyclic sleep period
    {"SN", "100"}, // number of sleep periods
    {"SM", "0"}, // sleep mode
    {"ST", "500"} // time before sleep
};
```

La procédure est la suivante: on fait une requête au module LoRa, si il est connecté alors il répond, si on a une réponse, on est coordinateur donc on change la valeur de “CE” à 1. On demande au XBee son adresse mac et on la met dans “NI” afin d'être sûr que deux cartes n'aient pas le même. Comme dit plus haut, c'est le “nom” avec la carte va se manifester sur le réseau. Enfin, on envoie tous nos paramètres stockés dans le tableau.

Voici l'automate responsable de cette tâche :



4.2 : Envoi et réception de trames :

On a vu comment on discutait avec le xBee lui-même, maintenant il faut communiquer à travers ! Pour cela, les données sont rangées dans des trames. Il existe plusieurs types de trames avec chacune une fonction. Celle qui nous intéresse est le type “receive packet” dans lesquels on revoit les données d'une autre ruche, et les “transmit request”s dans lesquelles on met les données à transmettre. Toutes les trames commencent par un octet “0x7E”, suivi de deux octets qui annoncent la longueur de la trame. A la fin, un checksum que l'on évoquera ensuite. Le reste de la trame est entièrement dépendant de son type, c'est pourquoi il faut un automate différent en fonction de la trame.

Notre programme possède une fonction qui prend une chaîne de caractères en entrée, et envoie la trame adaptée. Elle calcule la longueur de la trame, elle ajoute les infos nécessaires à la trame mais dont on ne se sert pas, la chaîne de caractère, puis le checksum. Ce dernier sert à s'assurer de l'intégrité du message transmis. Selon le site de digi (le fabricant de xBee), pour la calculer, il faut ajouter tous les octets de la trame sauf la longueur et le préfixe “0x7E”. Puis soustraire les 8 bits les plus faibles à “0xFF”. Voici cette fonction :

```
void xbee_send_string(uint8_t * string){  
    struct frame tx_frame = {0};  
    tx_frame.length = 0x0E + strlen(string);  
    invert_lsB_msB((uint64_t*)&tx_frame.length, 2);  
    tx_frame.type = 0x10;  
    tx_frame.id = 0x01;  
    tx_frame.address64 = 0xFFFF;  
    invert_lsB_msB((uint64_t*)&tx_frame.address64, 8);  
    tx_frame.address16 = 0xFFFF;  
    invert_lsB_msB((uint64_t*)&tx_frame.address16, 2);  
    memcpy(tx_frame.content, string, strlen(string));  
  
    uint64_t sum = 0;  
    uint16_t frame_length;  
    frame_length = 275;  
    uint8_t * ptr = (void *)&tx_frame+2;  
    for(uint16_t i=0; i<frame_length-2; i++){  
        sum += *ptr; // Il est plus facile de calculer le checksum avec des pointeurs  
        ptr++; // On l'a fait cette fois car on avait fait la méthode compliquée pour la  
réception  
    }  
  
    tx_frame.check_sum = 0xFF - (sum & 0xFF);  
  
    HAL_UART_Transmit(&huart5, "~", 1, 100); // "~" est égal à 0x75, la doc de xBee utilise les deux  
HAL_UART_Transmit(&huart5, (uint8_t*)&tx_frame.length, 2, 100);  
HAL_UART_Transmit(&huart5, &tx_frame.type, 1, 100);  
HAL_UART_Transmit(&huart5, &tx_frame.id, 1, 100);  
HAL_UART_Transmit(&huart5, &tx_frame.address64, 8, 100);  
HAL_UART_Transmit(&huart5, &tx_frame.address16, 2, 100);  
HAL_UART_Transmit(&huart5, &tx_frame.content_index, 1, 100);  
HAL_UART_Transmit(&huart5, &tx_frame.option, 1, 100);  
HAL_UART_Transmit(&huart5, &tx_frame.content, strlen(string), 100);  
HAL_UART_Transmit(&huart5, &tx_frame.check_sum, 1, 100);  
}
```

On note que la fonction `invert_lsB_msB` à été ajoutée parce que le driver UART du microcontrôleur envoyait les octets du dernier au premier, au lieu du premier au dernier. Un paramètre “caché” du microcontrôleur permet en fait de le faire sans notre fonction, mais on ne l'a trouvé qu'après coup.

Maintenant on veut aussi recevoir ces données. La difficulté est que cette trame ne possède pas de délimiteur de fin. C'est à dire qu'on doit lire sa longueur dès qu'on la reçoit, et compter le bon nombre d'octets ensuite pour l'analyser à la fin. Une fois cela fait, on peut vider les variables tampon et attendre le prochain préfixe "0x7E".

Pour faire cela, on a une interruption à la réception d'un octet, qui ajoute cet octet à un tableau. L'automate tourne dès qu'il y a une nouvelle valeur ajoutée (à la manière d'un registre à décalage). Cela permet de synchroniser la réception et la lecture. Sans ça, on risque de rater des octets.

La trame est à peu près la même, on y trouve aussi un checksum que l'on vérifie selon la règle énoncée par digi : Sommer tous les octets incluant le checksum, excepté le préfixe "0x7E" et la longueur. Si les 8 bits les plus faibles valent tous 1, alors il est peu probable que la donnée soit corrompue. (un checksum ne suffit pas à garantir l'intégrité, mais c'est un très bon indicateur).

L'automate est trop long pour le présenter ici mais vous pouvez le retrouver sur notre GitHub, la fonction `read_xbee()` déclarée dans main.c .

Une fois que la trame est confirmée par le checksum, on peut en extraire la chaîne de caractères qu'elle transporte.

On a créé une notation très simple à lire en C permettant de déterminer ce qu'on transmet. On détermine ce que c'est uniquement grâce à la première lettre.

Quand une ruche s'initialise, elle signale son fonctionnement au coordinateur avec la lettre "O" pour "online", suivie de son nom. Cela sert au coordinateur qui peut envoyer des instructions à une ruche précise selon les besoins de l'apiculteur. Il peut par exemple arrêter l'alerte d'une ruche avec la lettre "S" pour stop, si on inclut le nom de la ruche cible, seule celle-ci exécutera la commande, sinon, elles le font toutes. De la même manière, "V" pour demander un mode veille, "F" pour un échantillonnage fin, etc...

Étudiant auteur de cette partie : Antoine SERRY

4.3. TTN et le protocole LoRaWAN

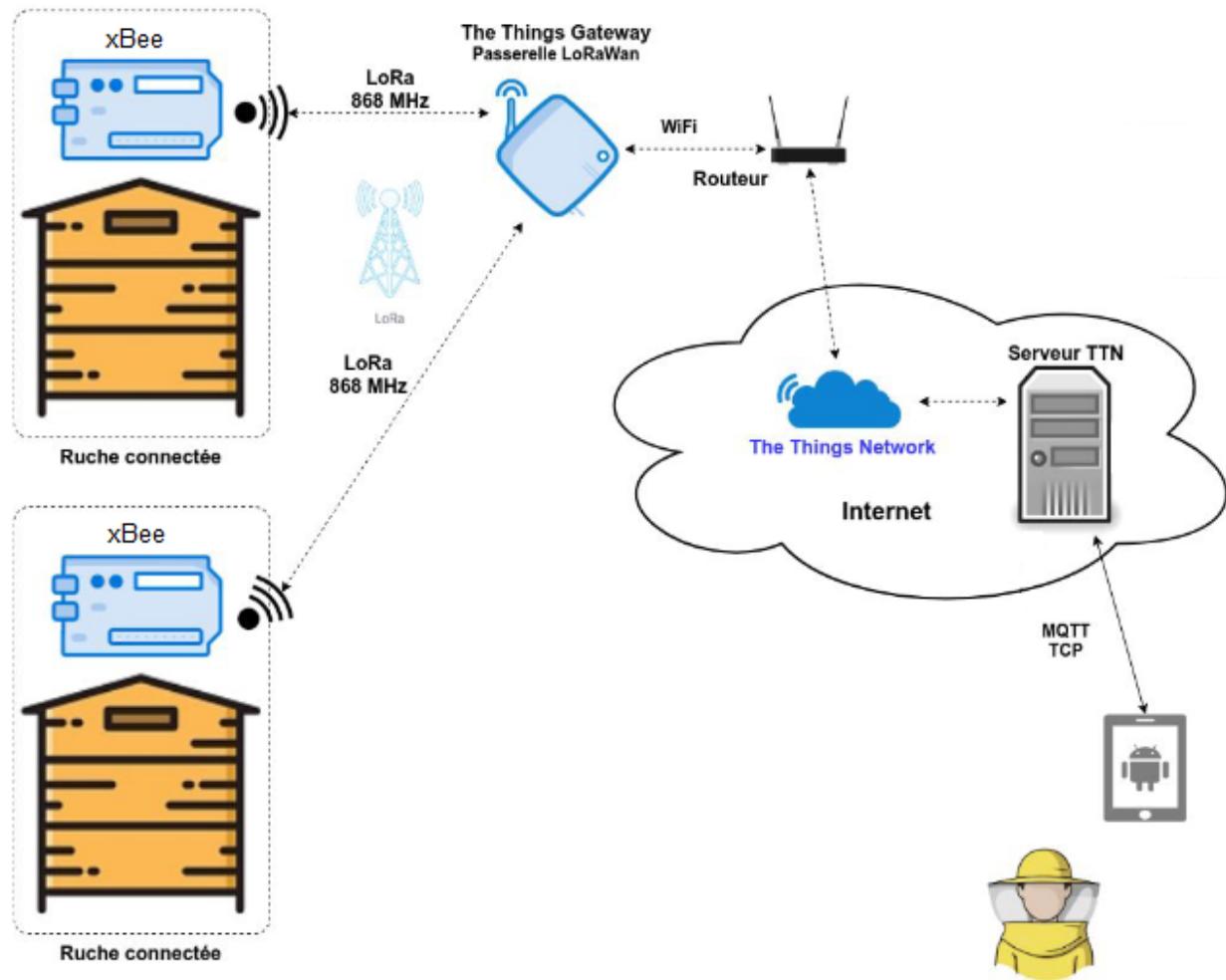


Le protocole LoRaWAN est un protocole de **télécommunication** permettant la communication à bas débit, par radio, d'objets à faible consommation électrique communiquant selon la technologie LoRa et connectés à internet via des passerelles, participant ainsi à l'**internet des objets** (Internet of things "IoT" en anglais). L'utilisation de ce protocole va de l'**agriculture** (ex: pour arroser des vignes en viticulture) au **monitoring industriel** ou encore dans les **villes intelligentes**. LoRaWAN est apparu grâce à l'achat d'une start-up grenobloise Cycléo par SemTech en 2012. Le protocole LoRaWAN permet grâce à sa partie physique LoRa, de connecter des capteurs ou des objets nécessitant une longue autonomie de batterie (pouvant se compter en années) mais dans le volume d'une boîte d'allumettes pour un coût réduit. LoRaWAN est l'acronyme de **Long Range Wide-area Network** et peut se traduire par "Réseau étendu à longue portée".

TheThingsNetwork est un **réseau communautaire** et **open-source** mondial pour l'internet des objets utilisant la technologie LoRa. Il est possible d'utiliser librement ce réseau mais il est également possible de l'étendre en installant des passerelles. TTN nous permet de mettre en œuvre le protocole **MQTT** pour recevoir et transmettre des données.



Dans notre système, nous utilisons une passerelle LoRa qui sera connectée au service **TheThingsNetwork** (TTN). Voici un schéma montrant le fonctionnement global :

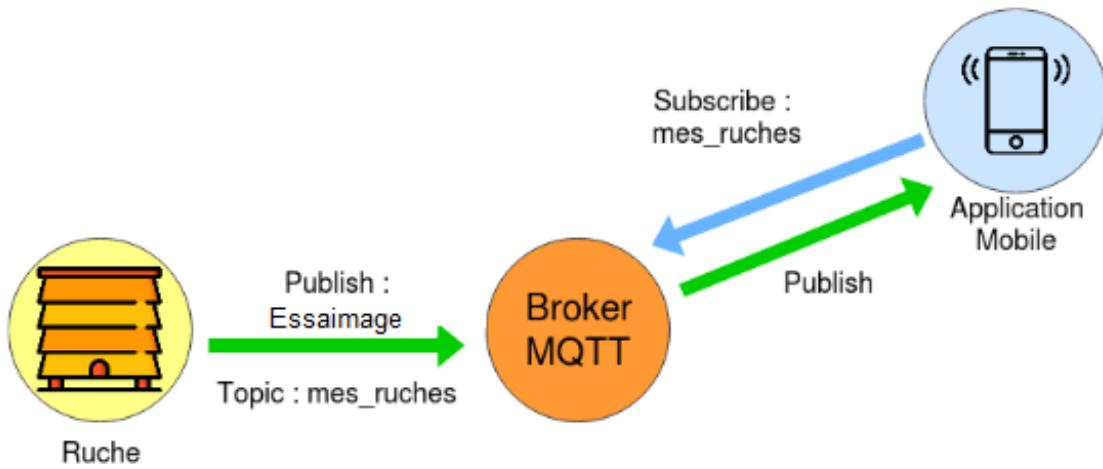


L'apiculteur dispose de plusieurs ruches, sur lesquelles seront posés des modules **xBee** fonctionnant avec la passerelle LoRa qui se chargera de récupérer les informations reçus des capteurs. Une fois ces informations reçues, la passerelle connectée au **routeur**, s'occupe de les envoyer sur **TheThingsNetwork** pour pouvoir ensuite recevoir une **notification** lors de la détection d'un essaimage sur le téléphone Android de l'apiculteur.

4.4. Protocole MQTT

MQTT est un protocole de messagerie basé sur le système de **publish/subscribe** qui utilise le protocole **TCP/IP**. Il est conçu pour les connexions avec des sites distants ou la bande passante du réseau est limitée, il est très utilisé pour l'**IoT** (l'internet des objets).

Ce protocole fonctionne avec un système de “**broker**” qui est un service permettant la mise en œuvre du flux de données, dans notre cas, il s’agit de **TheThingsNetwork**. Un système de “**topic**” qui va représenter le flux de données. Et enfin le “**publish**” peut servir à envoyer des données à la ruche par exemple, dans notre cas nous recevons des publish provenant de la ruche, nous utiliserons également le “**subscribe**” pour souscrire au flux et recevoir des données en cas d'essaimage.



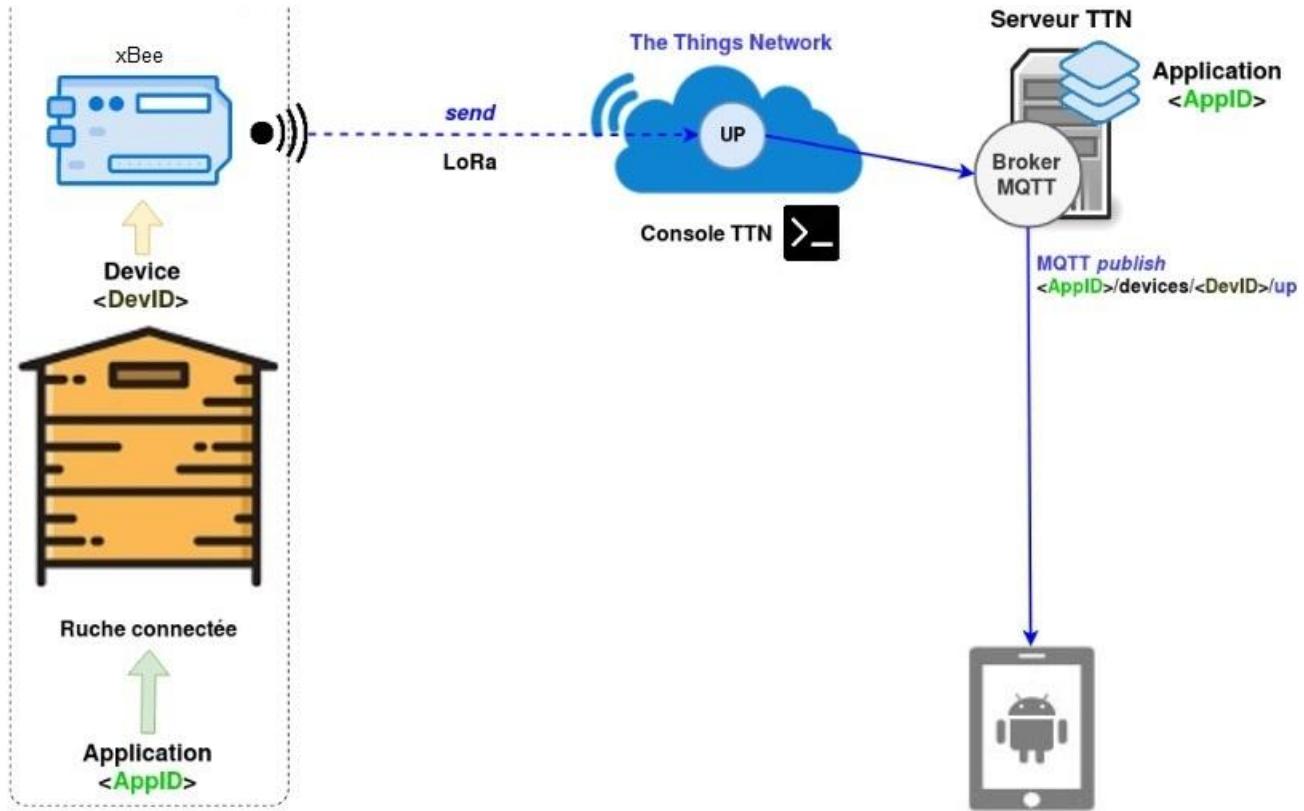
4.4. Application Android

Mon application Android doit envoyer une notification dès qu'un essaimage a été détecté. Pour le développement de cette application j'ai tout d'abord chercher à tester le fonctionnement du protocole MQTT, pour cela j'ai tout d'abord mis en place un serveur MQTT local avec **Mosquitto**. La commande ci-dessous une fois rentrée sur l'invite de commande windows nous permet de souscrire à un flux de données sur TheThingsNetwork, ainsi j'ai pu recevoir les données sur un ordinateur avant de me lancer dans la programmation de l'application en **Java**.

```
1 mosquitto_sub -h eu1.cloud.thethings.network -t
2 "test-id-123/devices/eui-70b3d57ed0045adc/up" -u "test-id-123"
3 -P "NNSXS.INFM2N4KRGDE78P03AQ...." -v
```

Pour la programmation de l'application, j'utilise **Android Studio** qui est un environnement de développement pour la création d'applications Android codé en Java. Le but est donc le même que lors de l'utilisation de **Mosquitto** souscrire à un topic TheThingsNetwork pour pouvoir recevoir des informations, en l'occurrence des informations qui vont nous permettre d'envoyer des notifications en cas d'essaimage.





Comment recevoir des notifications sur notre appareil Android lors de la réception d'informations sur TheThingsNetwork ?

Tout d'abord, comme nous pouvons le voir sur ce schéma, il y a la présence de différents identifiants qui peuvent nous permettre de souscrire à un topic. Le <DevID> ou “Device ID” est l'identifiant du device que nous pouvons créer sur TTN avec “+ Add end device”, nous pouvons voir ensuite l'ID du device s'afficher.

End devices (1)		Search by ID		+ Import end devices	+ Add end device
ID	Name	DevEUI	JoinEUI	Last activity	
eui-70b3d57ed0045adc		70 B3 D5 7E D0 04 5A DC	00 00 00 00 00 00 00 00	Never •	

Étudiant auteur de cette partie : Raoul PETREAN

Nous avons ensuite l'**<AppID>** qui est trouvable tout aussi facile lors de la création d'une application, à savoir qu'une application peut contenir plusieurs devices. Pour trouver l'**App id** il suffit de cliquer sur "Overview".



Il nous manque maintenant la "clé API" qui est une sorte de mot de passe pour pouvoir souscrire à notre topic, elles sont générées **aléatoirement** et sur demande en cliquant ici :

MQTT

The Application Server exposes an MQTT server to work with streaming events. In order to use the MQTT server you need to create a new API key, which will function as connection password. You can also use an existing API key, as long as it has the necessary rights granted. Use the connection information below to connect.

Connection credentials

Public address	<input type="text" value="eu1.cloud.thethings.network:1883"/> 
Public TLS address	<input type="text" value="eu1.cloud.thethings.network:8883"/> 
Username	<input type="text" value="ruche-123@ttn"/> 
Password	 <input type="button" value="Generate new API key"/> Go to API keys

Passons maintenant à la partie **Android Studio**.

Étudiant auteur de cette partie : Raoul PETREAN

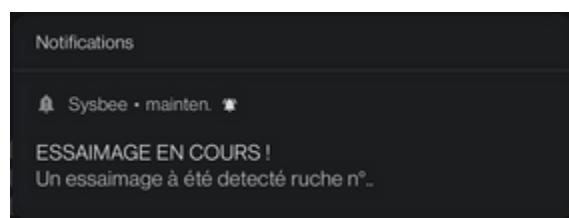
Un extrait de code en **Java** ici, montre le paramétrage de la commande en utilisant les **ID** précédemment cités. Dans le code “clientID” est en fait l’**App ID**, password l’**Api key** et le **device ID** se trouve dans le “subscriptionTopic” et commence par “eui”.

```
17     public MqttAndroidClient mqttAndroidClient;
18
19     final String serverUri = "tcp://eu1.cloud.thethings.network:1883";
20
21     final String clientId = "ruche-123@ttn";
22     final String subscriptionTopic = "v3/ruche-123@ttn/devices/eui-70b3d57ed0045adc/up";
23     final String username = "ruche-123@ttn";
24     final String password = "NNSXS.3R4VM7KLPG55TZAVQ7SLUBXGNK7NTEMS6WYI3Q.QYTH3FZJJZDWOF70H70QSL4TDUHYIMIC5HBRBH6B7P7ZRB7WTWEA";
```

Et voici la partie du code permettant de générer la **notification** en elle-même, avec les parties commentées qui explique les **différents paramétrage** d'une notification.

```
63     private void showNotification() {
64
65         createNotificationChannel();
66
67         // Création de la notification
68         NotificationCompat.Builder builder = new NotificationCompat.Builder(context, CHANNEL_ID);
69         // Icône de la notification
70         builder.setSmallIcon(R.drawable.ic_notification);
71         // Titre de la notification
72         builder.setContentTitle("ESSAIMAGE EN COURS !");
73         // Description
74         builder.setContentText("Un essaimage à été détecté ruche n°..");
75         // Priorité
76         builder.setPriority(NotificationCompat.PRIORITY_MAX);
77
78         NotificationManagerCompat notificationManagerCompat = NotificationManagerCompat.from(this);
79         notificationManagerCompat.notify(NOTIFICATION_ID, builder.build());
80     }
```

Nous obtenons à la fin cette notification qui permet d'avertir l'apiculteur lors d'un essaimage détecté sur l'une des ruches.



Étudiant auteur de cette partie : Raoul PETREAN

2 - Conclusion

_____ Pour conclure, nous sommes parvenus à réaliser un rendu de projet satisfaisant que nous estimons avoir complété à plus de 90% en utilisant les différents moyens techniques cités tout au long de ce rapport. Notre projet est fonctionnel et répond au cahier des charges fourni : le système est parfaitement adapté aux contraintes environnementales, le réseau permettant d'alerter l'apiculteur est fonctionnel, tout comme l'application destinée à l'utilisateur. Chaque partie attribuée a été traitée par les membres du groupe que ce soit de manière autonome ou en équipe, mais malheureusement un seul point n'a pas pu être rempli : nous ne sommes pas parvenus à établir une communication complète entre le microcontrôleur et le module LoRa de part la complexité de la tâche. Malgré cela, la démonstration de notre projet a été possible et est très satisfaisante. On peut donc affirmer que notre projet est un succès au vu de nos réalisations.

Étudiant auteur de cette partie : Ihsan DECIEUX

Bibliographie et Sitographie

- https://www.st.com/content/my_st_com/en/products/embedded-software/mcu-mpu-embedded-software/stm32-embedded-software/stm32cube-expansion-package/s/i-cube-lrwlan.license=1633089636873.product=I-CUBE-LRWAN.version=2.1.0.html
- <https://www.freecadweb.org/?lang=fr>
- <https://fr.rs-online.com/web/>
- <https://github.com/TheKnightWhoSaysNi/SysBee>
- <https://www.skyworksinc.com/-/media/SkyWorks/SL/documents/public/application-notes/an1131-layout-guide.pdf>
- https://wikifab.org/wiki/D%C3%A9ployer_une_passerelle_LoRaWAN_pour_The_Things_Network/en
- <https://fr.wikipedia.org/wiki/LoRaWAN>
- <http://tvaira.free.fr/dev/android/android-8.html>
- <https://www.thethingsnetwork.org/docs/>
- <https://www.thethingsnetwork.org/docs/applications-and-integrations/>

