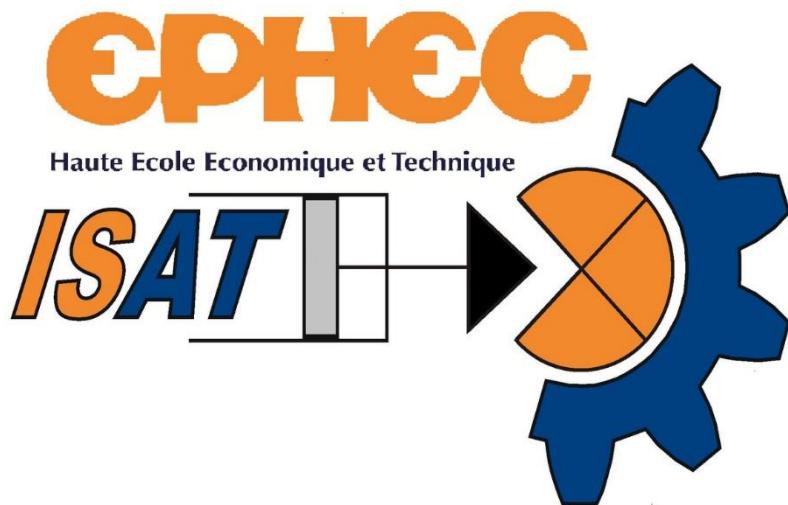
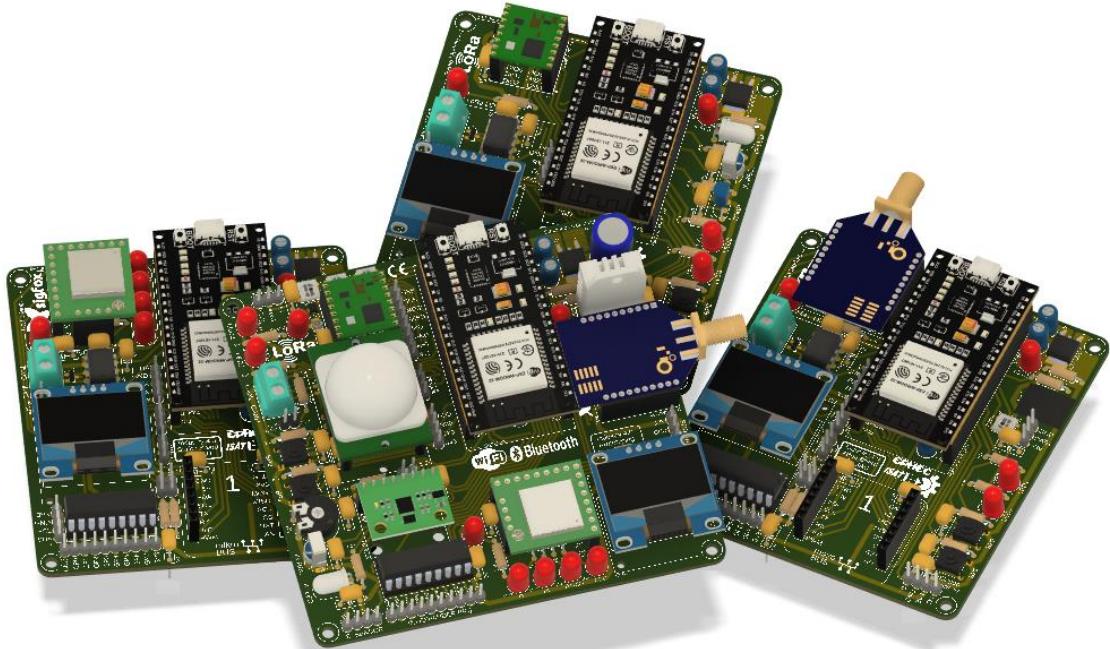


TRAVAIL DE FIN D'ETUDES

Haute Ecole EPHEC-ISAT – Département Technique
Bd Lambermont 17
1030 Schaerbeek



CONCEPTION DE CARTES ELECTRONIQUES POUR L'INDUSTRIE 4.0 COMMUNICATIONS SANS-FIL (LORA, ZIGBEE, SIGFOX, WI-FI, ...)



Travail de fin d'études présenté en vue de l'obtention du diplôme de
Bachelier en Automatisation

ETUDIANT : DI VENTI Davide
PROMOTEUR & RESPONSABLE : COSTA Emile

SECTION AUTOMATIQUE : 3AU
ANNEE ACADEMIQUE 2021-2022

Remerciements

Le développement et l'aboutissement de ce projet a pu être accessible grâce à mon entourage et à sa participation.

J'aimerais donc tout d'abord remercier mon professeur M. Emile Costa. Avant même avoir atteint le bloc 3 de mon bachelier, il a reconnu mon implication et ma motivation dans l'apprentissage du métier et m'a proposé un stage avec lui. Il a joué le rôle de mon maître de stage en interne ainsi que celui de mon promoteur. Il a été pour moi un pilier rempli de ressources et d'expériences sur lequel je pouvais compter dans le but de renforcer mes compétences.

Je remercie également tous les autres membres de l'équipe pédagogique de l'ISAT qui m'ont formé tout au long de mon cursus dans l'enceinte du département. Je pense aussi à M. JP. Cordier qui m'a permis d'être en ordre au niveau administratif en ce qui concerne mon stage/TFE.

Je m'adresse identiquement à toutes les personnes ayant indirectement impactés l'évolution de mon TFE : mes collègues, ma famille, les fabricants des PCB, les développeurs des outils tels que les librairies mis à notre disposition en open source et les personnes ayant pris leurs temps de créer des tutoriels.

Préface

L'adaptation du TFE au contexte sanitaire a été assez flexible dans la période de mon stage. En télétravail mes horaires ne sont pas fixes, j'étais le travail sur toute la journée de façon à combler mes 6 à 8h par jour. Ma principale préoccupation était que le contenu de mon stage soit freiné en raison du COVID. J'avais l'espoir que même dans le pire des cas, ce TFE pourrait très bien être effectué à mon domicile en cas de crise car le matériel n'est pas ce qui manque.

Malgré l'adaptation, un imprévu a eu lieu. Le 17 mars 2022, mon maître de stage a reçu une information de la part de l'expéditeur de notre PCB disant qu'il arriverait le 4 avril, l'équivalent de 20 jours de retard approximativement. On pouvait s'y attendre, probablement dû au conflit entre la Russie et l'Ukraine. Mais fort heureusement, le colis est arrivé à Bruxelles le jour même où l'information a eu lieu. Mais tel n'est pas le sujet de ce dossier.

On aborde de plus en plus la notion d'objets connectés dans le marché. Cette technologie devient de plus en plus exploitée pour la simple et bonne raison de sa dextérité à la fois informatique qu'électronique. Les domaines d'applications sont quasi infinis dans notre monde. Il y a quelques années, ces possibilités appartenaient à la science-fiction, et aujourd'hui, je me permets de les dompter devant vous.

Avant-propos

Vous vous en doutez, ce dossier aborde donc la notion de l'IoT (Internet of Things), autrement dit, l'Internet des objets ou l'interconnexion d'objets connectés intelligents. L'aspect de ce projet concerne plus précisément le nouveau concept du siècle dans le monde de l'industrie : l'industrie 4.0.

Ce nouveau concept reprend le principe de l'IoT, mais cette fois-ci de façon à créer un réseau privé au sein d'une entreprise. En plus de l'automatisation d'un procédé de fabrication d'un produit, l'industrie 4.0 offre l'option de programmer, de dépanner, d'améliorer, de lire et d'agir sur une machine située en terrain de fabrication en étant devant un PC à distance à domicile, tel un père. L'enjeu de l'industrialisation a toujours été de limiter les efforts physiques humains.

La notion de l'industrie 4.0 se repose sur plusieurs niveaux. Les données lues par les capteurs des machines de production sont acheminées et formatées de niveau en niveau afin de procéder à une acquisition intégrale de ces derniers dans un serveur. De ce fait, l'accès aux données tout comme l'accès à la gestion d'une machine est disponible à distance.

Ce concept s'enracine sur le principe de la communication. Des dizaines, voire des centaines de protocoles standardisés offrent ces fonctionnalités. Il s'agit de moyens de communications sécurisés et fiables pour le transfert d'information. La couche physique peut être à la fois sans-fil ou avec fil. Et c'est sur ces principes que mon projet s'est basé.

Table des matières

1. Introduction	7
1.1. Présentation de l'entreprise	7
1.2. Présentation du TFE.....	8
1.3. Conditions de travail	9
1.4. Cahier des charges	10
1.5. Invitation à la lecture.....	11
2. Rappels théoriques	12
2.1. Microcontrôleur & ESP32	12
2.2. Raspberry Pi 4.....	14
2.3. Étude théorique des différents protocoles de communications embarqués.....	16
2.3.1. Wi-Fi.....	16
2.3.2. Bluetooth.....	17
2.3.3. Zigbee	18
2.3.4. LoRa.....	19
2.3.5. SigFox	20
2.3.6. Infrarouge.....	21
2.3.7. SPI.....	21
2.3.8. I ² C	22
2.3.9. CAN	23
3. Développement du sujet.....	24
3.1. Situation initiale du projet.....	24
3.1.1. Développement des idées de la création des PCBs	24
3.1.2. Périphériques de tous les PCBs.....	26
3.1.3. Analyse des solutions d'un TP Indusrtie 4.0.....	32
3.2. Mise en œuvre de la solution hardware	34
3.2.1. PCB Multi-Protocole.....	34
3.2.1.1. Schema bloc	35
3.2.1.2. ESP32 Pinout	36
3.2.1.3. Schémas électroniques	37
3.2.1.4. Rendu final.....	39
3.2.2. PCB Mono-Protocole orienté Zigbee.....	41
3.2.2.1. Schema bloc	42
3.2.2.2. ESP32 Pinout	43

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

3.2.2.3.	Schémas électroniques	44
3.2.2.4.	Rendu final.....	45
3.2.3.	PCB Mono-Protocole orienté LoRa.....	46
3.2.3.1.	Schema bloc	47
3.2.3.2.	ESP32 Pinout	48
3.2.3.3.	Schémas électroniques	49
3.2.3.4.	Rendu final.....	50
3.2.4.	PCB Mono-Protocole orienté SigFox	51
3.2.4.1.	Schema bloc	52
3.2.4.2.	ESP32 Pinout	53
3.2.4.3.	Schémas électroniques	54
3.2.4.4.	Rendu final.....	55
3.2.5.	Anciennes versions des PCBs	56
3.2.5.1.	PCB Multi-Protocole (V1.0 à V2.0)	56
3.2.5.2.	PCB Mono-Protocole orienté Zigbee (V1.0 à V1.1).....	57
3.2.5.3.	PCB Mono-Protocole orienté LoRa (V1.0 à V1.1).....	59
3.2.5.4.	PCB Mono-Protocole orienté SigFox (V1.0 à V1.1)	60
3.2.6.	Supports imprimés en 3D.....	61
3.3.	Mise en œuvre de la solution software.....	62
3.3.4.	Explication de la programmation des périphériques & GitHub.....	62
3.3.5.	Explication de la solution du TP Industrie 4.0 : TP Zigbee2MQTT	62
3.3.6.	Explication de la solution du TP Industrie 4.0 : TP Multi-Protocoles	69
3.3.6.1.	Configuration initiale du Raspberry Pi 4	70
3.3.6.2.	Implémentation du Docker.....	71
3.3.6.3.	Container Portainer	72
3.3.6.4.	Container MQTT.....	73
3.3.6.5.	Container Node Red.....	75
3.3.6.6.	API proposé par Telegram	75
3.3.6.7.	Interactions entre l'utilisateur Telegram, les PCBs et le bot.....	77
3.3.6.8.	API proposé par Fire Base	79
3.3.6.9.	Interface utilisateur du Node Red	80
4.	Conclusion.....	81

1. Introduction

1.1. Présentation de l'entreprise

L'option technique de mon bachelier offre une multitude de postes dans le milieu industriel. La plupart des étudiants qui effectuent un stage en externe peuvent avoir la certitude d'être embauchés par la même entreprise après leur stage. Malheureusement, très peu de ces postes répondent à mes attentes. Je cherche un métier dans le domaine du développement de systèmes embarqués autour d'un milieu mécanique & informatique de préférence, telle la mécatronique ou la robotique.

Par conséquent, le choix de mon stage devient très compliqué avec un titre de bachelier. L'idée m'est donc venue de poursuivre mes études après l'obtention de mon diplôme pour avoir accès à plus de portes.

Puisque je compte poursuivre mes études, je voulais donc m'orienter vers une société proche du domaine électronique. Quelques idées me sont parvenues, dont un stage en interne. Mais avant même d'en parler à quiconque, un évènement a eu lieu à l'école. Mon professeur d'électronique, M. Emile Costa, m'a offert l'opportunité d'être son stagiaire. Le TFE proposé correspondait à mes attentes : le développement de cartes électroniques avec une couche informatique intégrée. J'ai donc immédiatement confirmé.

Le stage a donc eu lieu à l'ISAT, l'Institut Supérieur des Aumôniers du Travail. Il s'agit du même département où j'effectue mon bachelier. Fondé le 21 novembre 1894 à Schaerbeek, cette haute école technique a eu le mérite de former au mieux des techniciens de métiers en pénurie jusqu'à aujourd'hui. En 1995, l'EPHEC, fondée en 1969, un institut d'enseignement supérieur qui organise des formations de type bachelier, a rajouté l'ISAT en son sein pour supprimer son indépendance.

1.2. Présentation du TFE

Je développe mon TFE dans le cadre de mon stage. Il s'agit de plusieurs cartes électroniques intégrants plusieurs capteurs ainsi que plusieurs types de communications standardisées. Parmi eux, on retrouve le LoRa, Zigbee, Sigfox, Wi-Fi, Bluetooth, IR, UART, CAN, l'I²C et le SPI. Le microcontrôleur ESP32 centralise le tout.

La première partie du TFE consiste à la conception de circuits imprimés (PCB) et à la programmation de ces derniers. Les PCB enveloppent donc un maximum de communications normalisées en industrie 4.0 de sorte à pouvoir simuler de façon didactique le même processus qu'en entreprise.

Chaque PCB intègre une grande diversité de capteurs. On en retrouve de type accéléromètre, gyroscope, microphone, magnétiques à effet hall, capacitif, ultrasonique, NTC, LDR, PIR, et pleins d'autres encore... Les données de ces derniers sont la source utile de chaque message envoyé par communication lorsqu'on programme le tout.

Les algorithmes développés réuniront les périphériques concernés d'un PCB de façon à effectuer des acquisitions de données externes/internes et à exécuter les protocoles de communication en mode émetteur/récepteur de PCB en PCB à distance ou pas.

La deuxième partie du projet s'oriente plus spécifiquement dans l'informatique du réseau. En plus de la programmation individuelle de chaque périphérique, je suis amené à réaliser un TP (Travail Pratique) que les futurs étudiants en 3AU auront à reproduire. Outre le fait d'aborder la quasi-totalité des approches embarquées dans les différents PCBs, ce TP utilise aussi quelques principes d'un serveur.

A partir de Clouds, de notions d'API, de bases de données, et de quelques passerelles, les étudiants seront capables de développer une application permettant de gérer intelligemment à l'autre bout du monde les PCBs.

Localement, ces derniers (les PCBs) formeront réseau local communiquant entre eux via plusieurs protocoles de communications, notamment le LoRa, le Zigbee & le Wi-Fi.

Dans ce projet, plusieurs langages de programmations sont utilisés : C, JS, JSON, html, CSS, commandes bash sous WSL ainsi que des requêtes HTTP.

1.3. Conditions de travail

Le lieu de mon travail principal est un local nommé S2 LEO. Il s'agit d'un laboratoire électronique où l'on peut effectuer toutes sortes d'expériences. On y retrouve des alimentations, des oscilloscopes, des ordinateurs, des instruments d'ateliers mécaniques voire même un poste de création de PCB comme à l'ancienne.

Les outils informatiques qui m'ont été utiles dans le développement du TFE ont été les suivants :

Fusion 360

Conçu aux Etats-Unis, ce logiciel est commercialisé par Autodesk, lui-même fondé en 1982.

Il s'agit d'un programme initialement conçu pour la modélisation 3D de pièces mécaniques. Ce logiciel a la particularité d'intégrer également une facette de la conception électronique que j'ai beaucoup utilisé pour ce TFE (voir annexes 2 à 29).

La réalisation de circuits, de routages, de bibliothèques de composants, ainsi que la génération d'un fichier « gerber » du projet électronique sont entièrement disponibles dans ce puissant outil. La finition professionnelle que propose ce programme est comparable aux plus grands logiciels de conception électronique. La raison principale pour laquelle les électroniciens se tournent sur cet outil est qu'il est possible de générer un aperçu 3D du PCB, soit l'image tridimensionnelle de l'ensemble des plans électroniques précédemment créé. Très peu de logiciels proposent cette fonctionnalité en plus de toutes les autres déjà présentes dans Fusion 360.

Git & GitHub

Propriétaire de Microsoft depuis 2008, le Git & GitHub offrent un service aux développements de projets informatiques. Il s'agit littéralement du plus grand hébergeur de code au monde. Cet outil offre une gestion extrêmement bien organisée en ce qui concerne les travaux de groupe ou personnels. Se tourner vers cet outil supprime les efforts inutiles de mise en commun de code entre deux ou plusieurs versions créées par des utilisateurs différents d'un même projet.

Cet outil m'a permis de créer un dépôt dans lequel tous mes codes sources sont placés. Le principe de pull et de push me permet de mettre à jour mes modifications effectuées localement (GIT) et/ou en ligne (GitHub).

Visual Studio Code & PlatformIO

En parlant de codage, développé par Microsoft en 2015, ce Visual Studio Code a été la plateforme dans laquelle j'ai développé la plupart de mes algorithmes (voir annexes 32 à 37). Que ce soit en C, html, JS, CSS ou autre, cet outil puissant peut tout interpréter.

L'ajout du plugin PlatformIO dans Visual Studio Code m'a été nécessaire pour ajouter la fonction de développement embarqué. Cet IDE offre l'option de reconnaître le microcontrôleur branché en USB ajoutant ainsi la fonctionnalité de téléverser le script compilé. Il s'agit d'une extension ukrainienne conçue en 2014.

1.4. Cahier des charges

Introduction :

Ce Travail de Fin d'Etudes (TFE) portera sur le développement de matériel didactique articulé autour des objets connectés (IoT) qui sera exploité principalement dans l'unité d'enseignement « A311 – Industrie 4.0 » et destiné à des étudiants de 3ème année de Bachelier en Automatique.

Remarque : ce matériel pourra être exploité dans d'autres UE.

Corps du travail :

Le principe (point de départ) de ce matériel pourra être calqué sur la carte de développement pour ESP32 utilisée en BAC 3.

L'élément important de ce travail sera la communication entre les objets (ex : capteurs intelligents) et un serveur.

Dans un premier temps, il s'agira de la conception de matériel électronique (PCB) basée sur des modules ESP32 et exploitera les communications sans-fil suivantes :

- WLAN ;
- Bluetooth ;
- ZigBee ;
- LoRa ;
- Sigfox ;
- Autres.

L'architecture de ces PCB devra dans la mesure du possible (contraintes techniques) être assez semblable d'une technologie à l'autre.

Pour la partie réseau, les différentes solutions « Cloud » seront étudiées sous forme d'un comparatif où la notion de coût et la difficulté de mise en œuvre seront mis en évidence.

Les solutions Cloud seront :

- Amazon AWS Iot ;
- Azure Iot ;
- Google Cloud IoT.

Remarque : une approche au niveau d'un serveur (local) peut être envisagée et on pourrait utiliser un micro-ordinateur tel qu'un Raspberry Pi 4 ; mais dans ce cas-là, ne pas oublier les contraintes liées à un réseau sécurisé telles qu'une école, entreprise,

La notion d'API (Application Programming Interface) tels que REST, JSON, XML sera étudiée suivant les solutions (nœuds + serveur + Cloud) choisies.

Dans un deuxième temps, la mise en œuvre de la communication OPC UA sera développée sur une plateforme S7-1200 (serveur et/ou client) et d'autres clients (logiciel et ou matériel).

Une documentation technique (à destination des étudiants) la plus exhaustive possible sera également réalisée pour permettre une mise en service aisée de ce matériel pédagogique.

1.5. Invitation à la lecture

Ce dossier abordera plusieurs notions complexes de façon régulière. Quelques rappels théoriques sont mis en place pour mieux comprendre le sujet abordé. Plusieurs thèmes seront traités tout au long de ce rapport :

En premier temps se trouveront plusieurs études menées sur les différentes solutions matérielles à mettre en place pour mener à bien le projet. Une explication sur les circuits imprimés équipés de dizaines de périphériques, tous différents autant qu'ils sont, sera présente.

En deuxième temps, plusieurs études menées sur les différentes solutions logicielles auront lieu. En lien avec les communications standardisées embarquées dans le circuit imprimé (Wi-Fi, LoRa, Zigbee, SigFox, ...), l'idée serait de développer un TP (Travail Pratique) avec toutes sortes d'approches de l'Industrie 4.0 intéressantes à implémenter.

A nouveau, si vous avez très peu de notions dans le domaine, n'hésitez pas à vous référencer aux rappels théoriques. Ils seront présentés à partir de la page suivante afin de pouvoir plus facilement vous familiariser avec le contenu concerné de ce dossier.

2. Rappels théoriques

2.1. Microcontrôleur & ESP32

A moins que vous n'ayez jamais vécu dans un monde civilisé, vous avez sans doute tous déjà rencontré des feux tricolores aux quatre coins d'un carrefour. *L'un devient rouge, donc l'autre se met au vert, après quelques moments cela passe à l'orange puis au rouge à nouveau. Le piéton a appuyé sur un bouton pour accéder au passage, le cycle est interrompu...* Vous vous êtes sûrement déjà demandé comment cela fonctionne, mais vous n'avez jamais développé cette étude par vous-même. Je vous éclairerai donc sur ce principe de feu tricolore, le tout en introduisant la notion de microcontrôleur car il y a un lien conséquent entre ces deux derniers.

Vulgairement, un microcontrôleur est un tout petit composant électronique avec beaucoup de pattes. La quasi-totalité de ces pattes sont généralement appelées « I/O » (Input/Output ; Entrée/Sortie). Ces pattes peuvent être les yeux et les mains du microcontrôleur. En d'autres termes, elles peuvent lire des informations externes (le bouton des piétons) et/ou agir sur des périphériques externes (les lumières tricolores). Si nous devions reproduire les feux tricolores autour d'un microcontrôleur, cela signifierait que chaque lampe d'un feu est liée à une I/O (donc une patte), et chaque bouton de piétons est également lié à une I/O. Si le carrefour est composé de 20 lampes et de 4 boutons au total dans les feux tricolores, alors le microcontrôleur verra 24 de ses I/O liés aux divers éléments routiers. **Cette première étape est le câblage du microcontrôleur aux divers périphériques.**

Mais tout ceci n'explique pas encore comment rendre le processus fonctionnel. J'aborde donc **l'étape de la programmation du microcontrôleur**. La programmation permet de donner vie au système tout entier (le carrefour). En d'autres termes, il s'agit de l'étape où on introduit une logique au processus. On écrit littéralement dans le microcontrôleur toutes les étapes de fonctionnement voulues : *quelle lampe doit s'allumer en premier ? et en deuxième ? et dans quel cas le bouton agit sur les lampes ? etc...* Sans aller dans les détails, pour écrire les consignes dans le microcontrôleur, il n'y a pas 36 solutions, il faut le connecter à un ordinateur, via USB par exemple. Dans le cas où le microcontrôleur n'a pas de port USB, on utilise une alternative, on le câble dans un appareil nommé « programmateur » qui lui en a un. Arrivé à ce stade, pour pouvoir écrire dans le microcontrôleur, l'opérateur ouvre un programme IDE « d'environnement de développement intégré ». Dans ce dernier, le technicien commence à écrire des lignes de codes puis exécute une compilation et ensuite un téléversement vers le microcontrôleur. Le langage de programmation dépend des préférences du technicien. Vulgairement, il s'agit de langages pouvant être interprétés par l'humain et la machine. Dans la programmation, on est capable d'introduire des temporisations de quelques secondes (attente du feu rouge au vert), d'effectuer des lectures des différentes I/O (boutons appuyé ?), de forcer à 1 ou 0 les différentes I/O (lampe allumée, lampe éteinte) et bien plus encore...

En câblant et en programmant le microcontrôleur, on serait donc capable de faire une infinité de projets. Le domaine d'application ne s'applique pas qu'au feu tricolore, mais aussi à la conception de robots, CNC numérique, imprimante 3D, alarme domestique, voitures télécommandées, etc... Les principes restent les mêmes, un microcontrôleur est capable de gérer des entrées (boutons, capteurs, ...), et des sorties (lampe, moteurs, ...).

Vous savez maintenant ce que peut faire un microcontrôleur branché à plusieurs simples périphériques. Mais de quoi est-il composé pour exécuter de manière autonome les consignes prédéfinies dans son programme ?

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

J'ai précédemment cité le fait de compiler le code avant de le téléverser dans le microcontrôleur. Il s'agit d'une procédure qui permet de convertir le langage de programmation humain (C++, micro-python, ...) en une suite d'instructions interprétables par l'unité centrale du microcontrôleur : le microprocesseur ou CPU, l'unité de calcul. Ce script compilé est stocké dans une mémoire de type flash ou ROM pour ensuite être lu par le CPU. Les variables du code qui ont tendance à changer de valeurs à plusieurs reprises sont stockées dans la mémoire volatile RAM, particulièrement efficace pour la gestion de données temporaires. On remarque très vite qu'un microcontrôleur est un ensemble de composant autour d'un CPU. Il intègre également un quartz, un élément permettant de cadencer la vitesse d'instruction du CPU par seconde par le biais d'une horloge. Il est aussi équipé de périphériques permettant l'interaction au monde extérieur : interfaçage digitale I/O, PWM, ADC, DAC, I²C, SPI, UART, ... Chacun de ces périphériques internes est géré par le CPU grâce à plusieurs bus. Ce thème reste assez complexe, mais on peut toutefois maintenant réussir à imaginer le principe de fonctionnement interne d'un microcontrôleur.

Comme exemple concret, je me permets maintenant de présenter un microcontrôleur présent dans le marché et utilisé fréquemment autant par les entreprises que par les particuliers en raison de son avantage au niveau qualité/prix. Il s'agit de l'ESP32 d'Espressif, je l'ai utilisé pour le projet. Voici de quoi il est principalement équipé :

- **Un microprocesseur double cœur 32 bit avec une fréquence d'horloge à 240 MHz maximum :**
 - o 32 bit : L'accès à 2^{32} adresses de mémoires avec une largeur de bus de 32.
 - o Double cœur : Capacité de tourner deux différents programmes simultanément.
 - o Horloge : Cadence les instructions des CPU à une vitesse de 240 MHz maximum.
- **Une SRAM de 520 Ko :**
 - o SRAM : Mémoire vive statique conservant les variables temporaires du programme.
- **Une ROM de 448 Ko :**
 - o Principalement utilisé pour le stockage d'un programme et la lecture de celui-ci.
- **34 IO avec quelques PWM, ADC, DAC, touch sensor, ...**
 - o PWM : Une modulation de largeur d'impulsion permettant de varier la tension moyenne du côté d'une sortie IO
 - o ADC : Convertisseur analogique-digital 12 bit permettant de lire des variations de tension du côté d'une entrée IO et de les interpréter via un échantillonnage de 12 bit .
 - o DAC : Convertisseur digitale-analogique 8 bit permettant de créer une sortie IO variable analogique.
 - o Touch sensor : Déetecte les corps contenant des charges électriques, tel le doigt d'un humain
- **Wi-Fi & Bluetooth embarqué** (Expliqué en détails plus loin)
- **Protocole de communication SPI, I²C & UART** (Expliqué en détails plus loin)

2.2. Raspberry Pi 4

En considérant qu'à ce stade la notion de microcontrôleur vous est un peu moins vague, je trouve maintenant intéressant de poursuivre le thème en abordant le principe d'un micro-ordinateur, le Raspberry Pi.

Le choix de ce composant plutôt qu'un microcontrôleur est le fait que ce petit ordinateur monocarte a l'avantage d'effectuer des traitements d'images & vidéos efficacement, et bien plus encore. Ses performances de calcul dépassent considérablement celle d'un microcontrôleur. En effet, le Raspberry Pi intègre un processeur et non un microprocesseur.

Petite parenthèse, je n'ai pas abordé ceci, mais le principe de performance d'un microcontrôleur ou d'un ordinateur dépend littéralement de l'envergure des capacités de ce dernier. Pour faire un exemple, si mon microcontrôleur a une ROM de 448Ko cela signifie qu'en espace mémoire le programme ne peut pas dépasser cette limite. Si la RAM a une limite de 520 Ko d'espace mémoire, cela signifie que les traitements en cours d'activation ne peuvent pas dépasser cette limite. Autrement, le composant rencontrera des latences, des « lags » voir même des « bugs » et sera forcé de se redémarrer. Nous avons tous déjà rencontré des « bugs » lorsqu'on travaille sur notre ordinateur, c'est l'image d'une surcharge de données dans la RAM du PC, cela signifie que nous demandons à l'ordinateur de faire trop de choses en même temps.

Pour effectuer une comparaison avec l'ESP32, je présenterai également les performances du RPi (Raspberry Pi). Nous remarquerons très vite les différences conséquentes.

Mais avant toute chose, laissez-moi expliquer ce qu'est un Raspberry Pi. Je développerai les propriétés du dernier RPi apparu à ce jour (avant le RPi 400) : le Raspberry Pi 4. Il s'agit de la version que j'ai utilisée pour le TFE :

Le RPi 4 (Raspberry Pi 4) est donc un micro-ordinateur. Il est équipé de ports USB (pour y brancher des clefs USB, des claviers ou autre) ainsi que de sorties micro HDMI (pour y brancher des écrans). Et qui dit HDMI, dit carte graphique. Il intègre donc un chipset graphique (intégré dans son processeur) pour pouvoir gérer des écrans. Il s'agit d'un GPU VideoCore VI permettant le décodage matériel jusqu'à 4kp60. En tant qu'espace mémoire principale, un emplacement de carte SD est inclus (8Go minimum conseillé). Dedans, le firmware DOIT être uploadé pour pouvoir accéder aux ressources du RPi 4. Il s'agit d'un OS, Raspbian pour être plus précis, inspiré de Debian, un système d'exploitation, l'équivalent d'un Windows 11 dans un ordinateur par exemple. Sa mémoire vive SDRAM peut aller jusqu'à 8Go, ce qui est ENORME. J'ai personnellement choisi celui qui en détient 2Go. Avec un RPi 4 on a également la capacité d'ajouter une caméra via un connecteur. Sans aller plus loin, mais plutôt en restant dans le domaine de l'IoT (internet des objets), le RPi 4 a la particularité d'avoir également du Wi-Fi & Bluetooth intégrés. De plus, un port Ethernet est également disponible pour y placer des câbles RJ45 par exemple. Et enfin, étant donné que ce produit est principalement destiné aux amateurs de la technologie, il offre des broches I/O pour pouvoir agir sur des périphériques externes. Il est donc aussi équipé d'interfaçages digitales I/O, PWM, ADC, DAC, I²C, SPI, UART, ... presque comme l'ESP32.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Nous comprenons maintenant toutes les qualités qu'à un Raspberry Pi comparé à un microcontrôleur tel qu'un ESP32. Pour les plus curieux, voici une image un peu plus détaillée :

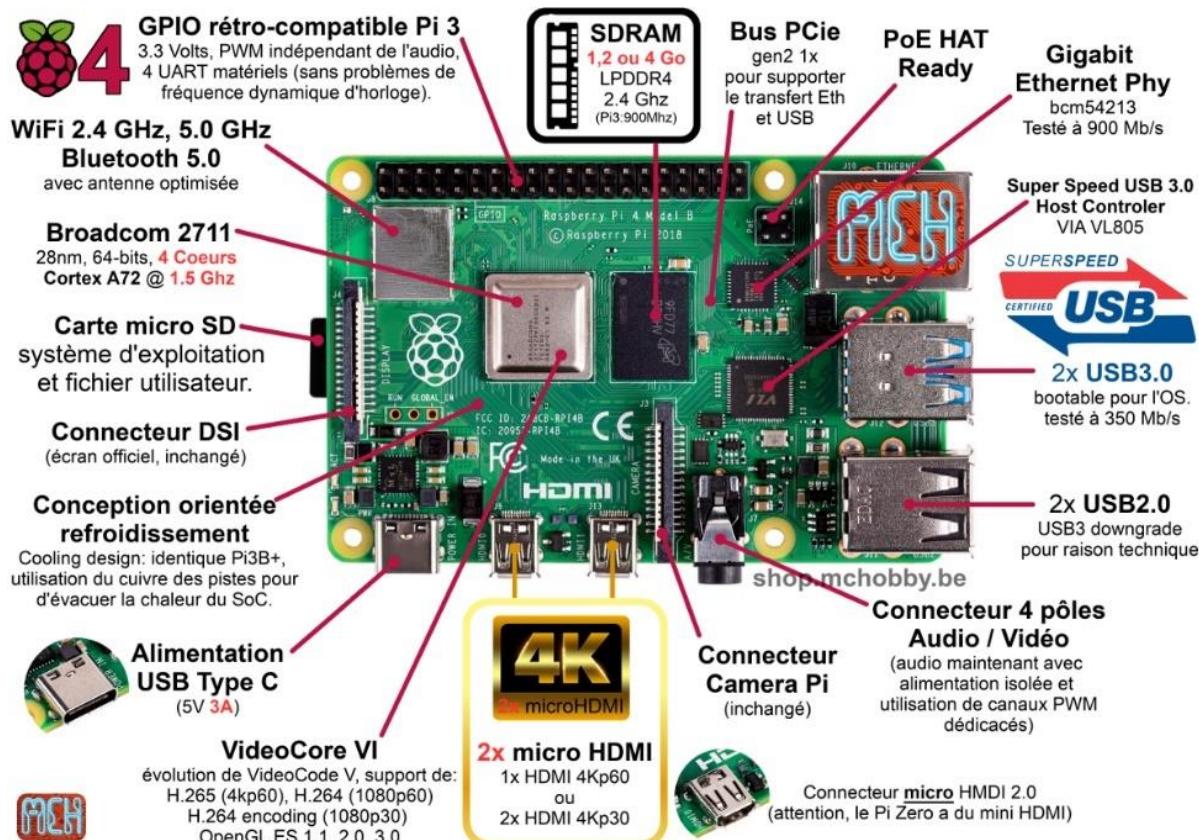


Figure 1 : Raspberry Pi 4

Figure 2 : Raspberry Pi 4

Si vous vous en souvenez, j'avais précédemment évoqué le fait qu'on choisit généralement la solution RPi pour ses raisons de performance en ce qui concerne le traitement d'images & vidéos. Ce n'est pas totalement vrai, car ce qui attire les consommateurs est plutôt le fait qu'il soit si polyvalent et si puissant. Mais alors, pourquoi ne pas supprimer les microcontrôleurs et n'utiliser que des RPi, vous me direz ? Eh ben la réponse est son prix. Le microcontrôleur ESP32 coûte environ 1 à 10€ l'unité, alors qu'un RPi 4 dépasse les 50€ et peut atteindre jusqu'à 250€. Alors avant d'acheter le matériel, il faut analyser les ressources que notre projet utilisera et ensuite se tourner vers la solution plus adaptée, RPi ou microcontrôleur.

La raison pour laquelle j'ai utilisé le RPi 4 n'a pas été celle de ses performances en ce qui concerne le traitement d'images & vidéos, mais plutôt celle du fait qu'un RPi puisse interagir en tant que serveur.

2.3. Étude théorique des différents protocoles de communications embarqués

Le PCB est en mesure d'exploiter plusieurs communications sans-fil. A ce jour, il intègre 9 types de communications différentes, dont 5 précisées dans le cahier des charges. On y retrouve le Wi-Fi, le Bluetooth, le ZigBee, le LoRa, le SigFox, le CAN et l'infrarouge. Comme la plupart des microcontrôleurs, le PCB est aussi capable d'exécuter les protocoles I²C & SPI.

2.3.1. Wi-Fi

Le **Wi-Fi** est un protocole de communication sans-fil utilisant les bandes passantes de 2.4GHz à 5GHz. Mis au point en 1997, cette technologie est principalement utilisée pour la connexion de plusieurs appareils, tel qu'un ordinateur, un smartphone ou autre, avec Internet.

En d'autres termes, l'accès aux banques de données d'internet est possible à partir d'un appareil embarquant cette fonctionnalité sans être connecté par câble.

Le Wi-Fi peut également être utilisé localement, c'est-à-dire à partir d'un appareil vers un autre sans exploiter Internet. Mais cette application est beaucoup moins fréquente, car d'autres communications sont plus adaptées (prochainement expliqué).

En bref, lorsqu'un appareil sans-fil souhaite émettre une information à Internet, tout se joue en premier temps sur sa carte réseau sans-fil. Cette carte est généralement intégrée dans la carte mère de l'appareil, mais dans le cas contraire, un adaptateur tel qu'un dongle en USB en fera pleinement l'affaire. Cet équipement traduit le format du message à envoyer utilisé par l'appareil en un signal radio.

En deuxième temps, le routeur ou la box reçoit le message émis sous le protocole Wi-Fi par l'antenne de cet adaptateur afin de le décoder. Une fois ceci effectué, ces informations sont transmises par câble Ethernet afin d'en informer à Internet.

De l'autre côté, lorsque l'appareil reçoit un message de la part d'Internet, le processus inverse a lieu. Ainsi, grâce au Wi-Fi, l'interconnexion de tous les appareils connectés est possible.

Dans les PCBs Industry 4.0 que j'ai réalisés, le module intégrant cette fonctionnalité sans-fil est l'ESP32. En plus d'être l'équipement principal pédagogique du PCB, il a l'avantage d'équiper non seulement cette fonctionnalité mais également celle du Bluetooth...



Figure 3 : ESP32

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

2.3.2. Bluetooth

Le **Bluetooth** est également un outil de communication sans-fil, mais cette fois- ci, adapté uniquement à une connexion d'échange de données numériques entre appareils électroniques localement.

Comme le Wi-Fi, la bande de fréquence utilisée est autour de 2.4GHz. Apparue en 1998, cette technologie a toujours eu le but de prendre en charge la communication d'appareils sans s'incruster dans un réseau externe.

On retrouve respectivement 2 types de technologie :

Le Bluetooth Classic : Ce protocole utilise une radio à faible puissance et occupe 79 canaux dans la bande de fréquence ISM (industrielle, scientifique et médicale).

Il prend en charge la communication *point à point*. Il s'agit d'une topologie ayant deux hôtes ou nœuds se communiquant entre eux uniquement, tel un serveur et un client.

Le Bluetooth Low Energy : Il est conçu principalement pour un fonctionnement à très faible consommation. Selon les cas d'utilisation, Bluetooth Low Energy consommerait environ 100 fois moins que Bluetooth Classic. Il occupe 40 canaux dans la bande de fréquence ISM.

En contrepartie, BLE (abréviation de Bluetooth Low Energy) se doit d'échanger des petites quantités de données comparées à la solution Classic.

En plus de prendre en charge la topologie *point à point*, il peut répondre aux solutions de *diffusion* et de *maillage*. Et tout ceci permettant de prendre en charge la création de réseaux d'appareils à grande échelle suivant le protocole Bluetooth.

La solution *diffusion* du BLE aborde la notion de serveur et de nombreux clients. Quant à celle du maillage, tous les appareils sont connectés, l'équivalent d'une connexion de plusieurs à plusieurs.

Comme cité précédemment, le module ESP32 a la capacité d'exploiter cette technologie. Il peut gérer le protocole **Bluetooth Classic** et **Bluetooth Low Energy**. Mais à ce jour, étant donné que ces technologies sont plutôt récentes, des exemples de code source sont très limités sur le net. Seul la topologie point à point sera l'application utilisée pour le TFE.

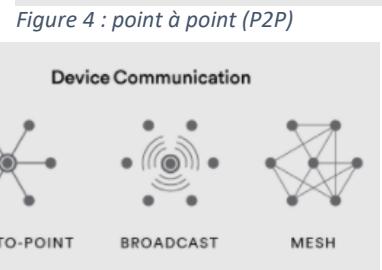
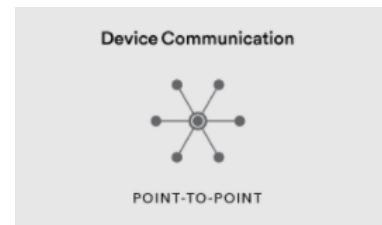


Figure 4 : point à point (P2P)

Figure 5 : P2P, Diffusion & Maillage

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

2.3.3. Zigbee

Le **ZigBee** et le **Xbee** sont très souvent confondus par les amateurs. Ces deux termes ont une différence à ne pas nier malgré leurs prononciations du moins fort semblables.

Le Zigbee est un protocole de communication. Il apparaît sous forme radio sans fil également à 2.4GHz. Il s'agit d'une marque déposée de Zigbee Alliance. Adapté à un réseau maillé, ce protocole est couramment utilisé dans la domotique, l'énergie intelligente, la détection et l'automatisation industrielle.

En parlant de réseau maillé, chaque réseau Zigbee attribue différents rôles aux nœuds qu'il compose. On en retrouve 3 :

Le coordinateur : Il s'agit d'un appareil Zigbee configuré pour qu'il soit le responsable du réseau maillé. Il distribue les adresses, il observe en temps réel le réseau tout en le sécurisant. Un réseau Zigbee ayant plus d'un coordinateur ne peut pas exister.

Le routeur : Le routeur est également un appareil Zigbee que l'on peut configurer comme tel. Le routage, à proprement parlé, est l'image d'un messager. Pour les appareils trop éloignés ne pouvant pas envoyer des messages par eux-mêmes, ce routeur fait office de passerelle. Le routeur peut également agir comme périphérique final...

Les périphériques finaux (endpoints) : Un périphérique final peut rejoindre un réseau et agir comme récepteur et/ou émetteur d'informations. Ces informations contiennent des charges utiles. Par exemple, la température ambiante d'une pièce, l'humidité d'une autre pièce, etc...

Il est intéressant de préciser que le Zigbee est un standard, tout comme le Bluetooth pour faire un exemple. En d'autres termes, si un casque Apple utilisant Bluetooth peut être lié à un smartphone Samsung, alors un interrupteur d'éclairage Zigbee de Xiaomi peut communiquer avec des lampes Zigbee de Phillips.

En revanche, le Xbee est un ensemble de module radio, propre à Digi International. Cette famille intègre des points communs : ceux d'avoir des outils et logiciels (notamment XCTU) de prise en charge mais aussi ceux de supporter un ensemble de groupe de protocole, notamment le Wi-Fi, le Zigbee et j'en passe.

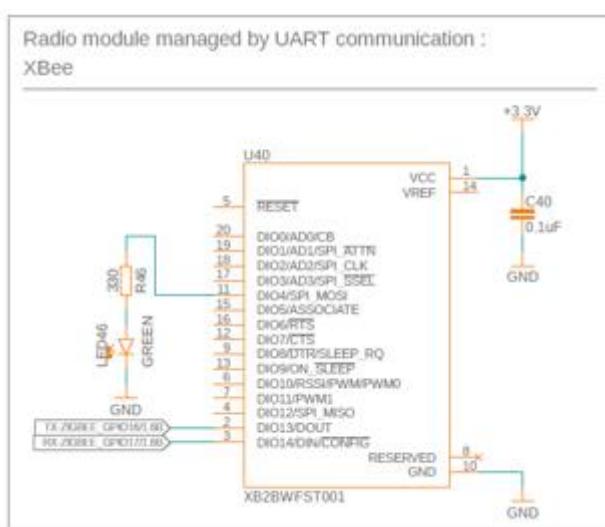


Figure 6 : XBee SCHEMA

Dans les PCBs Indusrt 4.0 embarquant la fonctionnalité Zigbee, j'ai utilisé des modules XBee 2SC.

Ces modules sont également des microcontrôleurs, mais je n'exploite que leur capacité à effectuer des communications Zigbee.

Depuis l'ESP32, comme représenté ci-contre, (voir annexes 5 et 11) ce XBee 2SC est commandé en UART. Il s'agit d'une communication série prochainement expliquée.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**2.3.4. LoRa**

LoRa est le diminutif de Long Range Radio. Il s'agit d'une technologie sans fil créée dans l'idée où il y a une communication entre deux machines (M2M : Machine to Machine) ou dans le monde de IoT.

Sous forme d'émetteur/récepteur, les modules de communications radio embarquant le protocole LoRa peuvent avoir une portée allant de 2km (milieu urbain) à 15km (banlieue). En Europe, la bande passante autorisée est celle 868MHz. Il s'agit d'une bande de fréquences pouvant être utilisée librement, utilisée également par l'ISM (Industrie, Science, Medical).

La solution LoRa a été apportée par Lora Alliance initialement formée pour standardiser LPWAN, le diminutif de Low Power Wide Area Network. Plusieurs membres de cette alliance, dont Proximus et Cisco, ont participé à la création du protocole LoRa, suivie de LoRa WAN.

LoRa WAN est le diminutif de Long Range Wide Area Network. Ce protocole offre la possibilité de communiquer à distance sur un de nos émetteur/récepteur placé n'importe où dans le monde. Il faut qu'il y ait des infrastructures LoRaWAN à portée des points de terminaison. Ces infrastructures intègrent des antennes à longue portée qui distribuent le service LoRaWAN. Malheureusement, il s'agit d'une solution apparue assez récemment et manque donc encore assez d'infrastructures dans le monde.

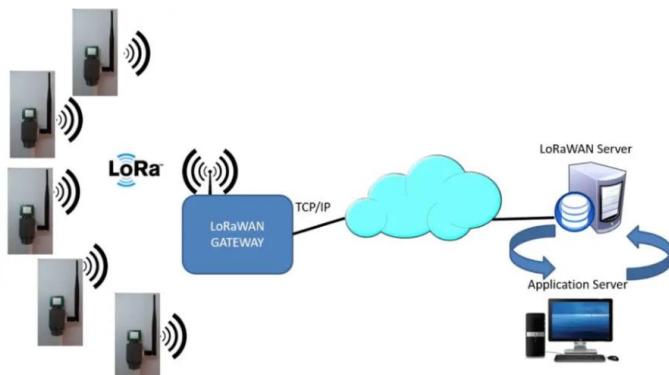


Figure 8 : LoRaWAN Processus

Dans les PCBs Industry 4.0 qui inclut la fonctionnalité LoRa, c'est le microcontrôleur émetteur /récepteur Lora RFM95 qui est utilisé.

Entre l'ESP32 et cet émetteur /récepteur LoRa, il y a une communication SPI qui se charge de signaler au RFM95 d'envoyer un signal LoRa ou d'en recevoir. Le SPI sera prochainement expliqué.

Comme représenté ci-contre, deux autres broches du RFM95 sont câblées (voir annexes 5 et 18). On y retrouve son D0 et son RST. La gestion de la broche RST est essentielle lorsque l'ESP32 essaie d'initialiser le module LoRa. Quant à la broche D0, elle se charge de créer un événement, telle une interruption pour le rappel de réception.

Indirectement, le LoRaWAN qui accueille les différents émetteurs/récepteurs, est connecté au réseau Internet par le biais d'une communication TCP/IP vers ses propres serveurs LoRaWAN. Ainsi, depuis nos cellulaires ou PC, en Wi-Fi ou Ethernet, il nous est possible d'accéder aux ressources acquises par notre réseau LoRaWAN équipé d'émetteurs/récepteurs épargnés dans tout le monde comme représenté ci-contre.

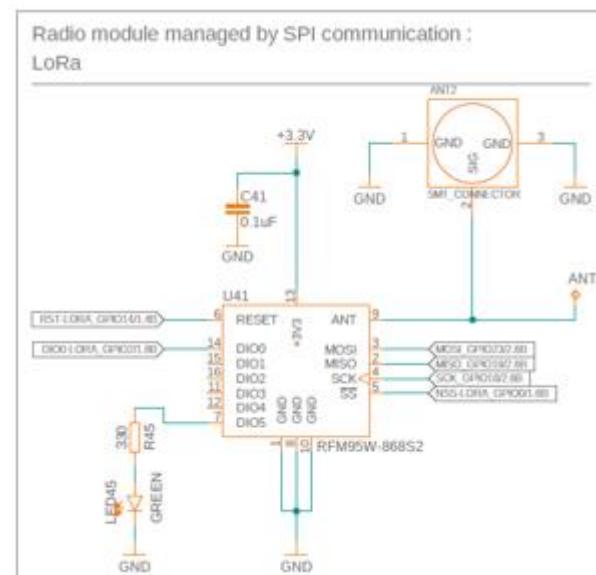


Figure 7 : LoRa SCHEMA

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**2.3.5. SigFox**

Identiquement au LoRaWAN, le SigFox peut se comparer au réseau cellulaire de nos smartphones nous permettant de nous connecter sur Internet. Le SigFox est conçu pour être plus précisément adapté aux appareils IoT. Il occupe également la bande passante ISM de 868MHz.

SigFox est configuré de telle sorte qu'il utilise moins d'énergie comparé au Wi-Fi ou à la 4G. En effet, on le surnomme 0G. En contrepartie, nous sommes limités sur les données, tout comme le LoRaWAN. Pour parler de chiffre, un total de 12 octets par message avec 144 messages par jour nous est permis.

Comme le LoRaWAN, la mise en disposition de ce genre d'émetteur/récepteur est exploitée dans le cas où il n'est localement pas possible de se connecter sur Wi-Fi ou d'utiliser la 4G.

Lors de l'achat d'un nouvel appareil SigFox, un identifiant sur l'étiquette du produit est placé. Cet identifiant est repris lorsque nous ajoutons au réseau SigFox ce nouvel appareil. Ainsi, une fois incrusté dans le réseau, le module SigFox est prêt à l'emploi.

Rendez-vous donc dans le site officiel de SigFox pour introduire cet ID, suivi du PAC (également disponible depuis l'étiquette du produit). Il vous sera ensuite demandé de créer un compte, et le tour est joué. Pour accéder aux données publiées par le module SigFox il faut se rendre sur le site <https://backend.sigfox.com/auth/login>.

Time	Seq Num	Data / Decoding	LQI	Callbacks	Location	
2022-03-29 21:17:31	33	24915e counter: 36 rv: -111 elegoo: 94				Vous y retrouverez toute sorte d'interfaces. Celles qui correspondent aux données publiées se trouvent dans Device -> Messages comme représentés ci-contre.
2022-03-29 21:16:26	32	254471 counter: 37 rv: 68 elegoo: 113				Comme exemple, sur base des informations publiées, il nous est possible de créer des notifications par mail reprenant ces données.
2022-03-29 21:15:17	31	26097b counter: 38 rv: 9 elegoo: 123				Ce backend est très polyvalent en termes de traitement de données.
2022-03-29 21:14:09	30	27d57b counter: 39 rv: -43 elegoo: 123				

Figure 9 : SigFox Backend datas

Le module SigFox que j'ai utilisé est le modèle BRKWS01. Il est uniquement disponible chez le fournisseur ydom de France.

Ce module SigFox communique avec l'ESP32 en UART également, comme le module Zigbee.

Une particularité de ce microcontrôleur est qu'il intègre des broches indiquant le statut de son CPU, son antenne ainsi que de sa communication UART. Comme représenté ci-contre, j'ai profité de ces broches pour placer des LEDs m'indiquant ainsi tous ces états (voir annexes 5 et 25).

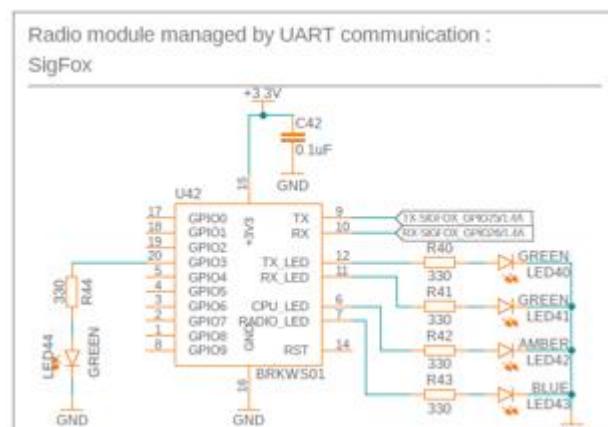


Figure 10 : SigFox SCHEMA

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

2.3.6. Infrarouge

L'infrarouge en soi n'est pas un protocole de communication, il s'agit plutôt d'une couche physique que l'humain utilise pour envoyer des données sous une fréquence porteuse. Car cette lumière, aux yeux de l'humain, est invisible, c'est une question de confort. Ces données peuvent être envoyées depuis un protocole, notamment le NEC (38KHz), le Sony (40kHz), etc...

La longueur d'onde est trop grande pour être perceptible par l'œil d'un humain. En effet, elles sont comprises entre 700 nm jusqu'à 1mm. Cette lumière, ou plutôt ce principe de rayonnement infrarouge, a été découvert en 1800 par William Herschel.

Bref, dans les PCBs Industry 4.0, les modules permettant cette fonctionnalité infrarouge ne sont rien d'autre qu'une LED IR, et d'un récepteur infrarouge. La LED IR émet une lumière comme une LED banale en silicium, sauf que la longueur d'onde de cette lumière correspond à de l'infrarouge.

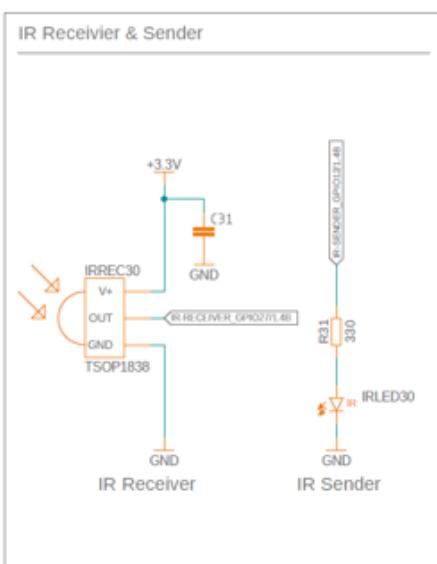


Figure 11 : IR SECHEMA

En revanche, du côté du récepteur infrarouge, toute une architecture est intégrée.

La détection de lumière se fait par le biais d'une photo diode en premier temps. Ensuite un processus de filtre de passe bande se charge de sélectionner la bande passante correspondant à la fréquence de communication. Il y a ensuite la démodulation puis l'amplification du signal de données.

Les signaux d'émetteurs et récepteurs infrarouges sont connectés par des simples broches digitales de l'ESP32 comme représenté ci-contre (voir annexes 3, 4 et 18).

Les protocoles de communication seront simulés par le biais de bibliothèque dans le code du microcontrôleur.

2.3.7. SPI

Le SPI est une interface basée sur le principe de la communication série. Un bus contenant 4 connexions est requis au minimum pour une application de type SPI.

La connexion SCK cadence le débit des données depuis une fréquence carrée. Elle peut atteindre l'ordre de quelques MHz. Ces données sont transportées unidirectionnellement dans les connexions MOSI et MISO comme représenté ci-contre : Quant à la connexion SS_x (où x=1), il s'agit du chip select, ou du slave select (c'est identique).

Plus il y a d'esclaves, plus de broches digitale SS_x devront être sacrifiées. Si le bus contient 3 périphériques SPI esclave, alors chacun d'entre eux occupera un SS personnel.

Ce SS est géré par le maître, il permet d'appeler/sélectionner l'esclave devant être interrogé ou informé. Ainsi, toute la communication SPI se canalise du maître à un de tous les esclaves (un à la fois).

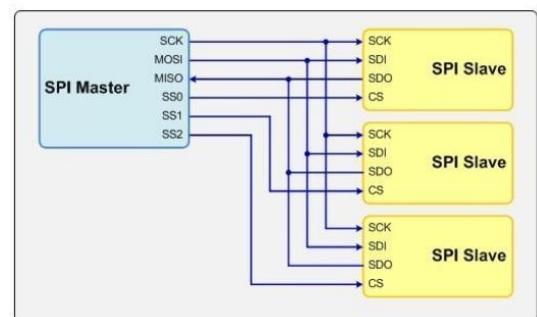


Figure 12 : SPI Processus

2.3.8. I²C

L'I²C est un protocole plus lent que le SPI. Son avantage est qu'il intègre dans son bus que deux connexions, SCL & SDA :

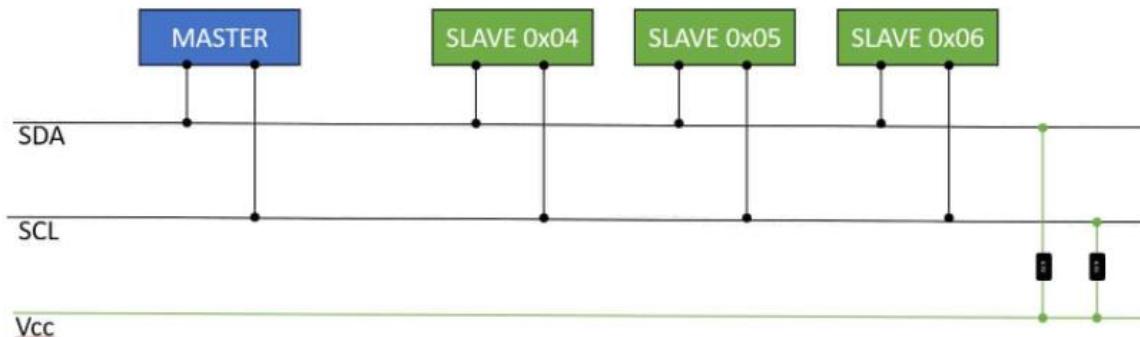


Figure 13 : I²C Processus

Comme le SPI, ce protocole est pré-disponible dans la majorité des microcontrôleurs. Pour éviter tout interférence avec le monde extérieur, des résistances sont requises, comme représenté ci-dessus. Elles tirent le niveau logique du bus vers le haut (3.3V dans mon cas). Ainsi, lorsqu'aucune communication n'a lieu dans le bus, nous sommes certains que le potentiel s'élève à 3.3V dans le bus plutôt d'avoir des valeurs inconnues venant de l'électricité statique (par exemple).

La ligne SCL est l'horloge du bus. Elle va cadencer les données passant dans la ligne SDA. Ces données sont bidirectionnelles.

Afin que le maître sélectionne un l'esclave pour l'interroger ou l'informer, il fait appel à son adresse I²C. Dans la trame se trouve principalement cette adresse suivie de la donnée utile. Cette trame voyage dans toute la ligne SDA, et lorsque l'esclave se rend compte qu'il a été appelé, il réagira.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

2.3.9. CAN

Le système CANbus est très répandu dans le domaine des automobiles. Il est extrêmement fréquemment utilisé à travers la communication des différents périphériques d'une voiture. Il émet un bus de données half-duplex, avec un support physique de paires torsadées ayant leurs extrémités bouclées par des résistances de 120Ω . On dit en général que le CAN Bus est capable de supporter un maximum de 20 nœuds avec une basse fréquence de flux de données (<125 kb/s). Plus la vitesse est grande (Max 1Mb/s), plus la longueur du CAN-Bus tolérée est petite.

Avant la modernité des automobiles, aucun système de communication n'était inclus à travers le réseau d'un véhicule. Chaque élément était lié au DashBoard de la voiture : les lumières, l'air conditionnée, la direction assistée, les freins, les sièges, etc... Aucun protocole, trop de fils électriques, un vrai casse-tête, bref une très mauvaise manie de raccordement de l'époque.

L'utilisation de ce protocole dans une automobile est une des meilleures solutions par rapport aux autres protocoles standardisés. Il a l'avantage d'être très efficace lorsque la longueur de bus est inférieure à 50m. Plus la distance est petite plus le flux de données autorisé est grand, ce qui est idéal dans un véhicule, car la longueur filaire ne dépassera pas ce seuil inférieur.

Ce qui est intéressant avec la communication CAN, c'est qu'il s'agit d'un réseau d'égal à égal. Il n'y a ni de maître, ni d'esclave. L'émission d'une trame est très organisée. En effet, le protocole analyse le bus afin de savoir s'il y a une trame en cours ou pas avant d'en émettre une nouvelle. Il y a également la notion de l'arbitrage dans le protocole. Il s'agit principalement d'un procédé qui définit selon des nœuds en compétition qui émettent simultanément un message la priorité de l'un d'eux. Le nœud ayant la plus grande priorité sera celui qui aura le privilège d'émettre son message. Quant aux autres, ils réessaieront d'émettre leur trame lors d'une autre tentative.

Le système CANBus ne cessera pas de croître dans le monde technologique. Il a actuellement 2 versions du protocole, la V2.0 publiée en 1993 embarque en plus de la précédente un nouveau format de trame, celle étendue de 29bits. De plus, le débit a pu atteindre les 1Mb/s dans les couches physiques en 1995. La transmission de données à faible consommation s'est également très développée avec le temps. Vers les années 2005 des nouvelles normes ont vu le jour également : celle CAN high-speed (1 Mb/s) et celle CAN low-speed (125 kb/s), actuellement utilisées dans les automobiles d'ailleurs. Elles permettent de différencier les éléments d'une voiture devant avoir une communication instantanée/urgente (freins, suspensions, ABS, Gestion moteur, éclairage, ...), plutôt que celles négligeables/non essentielles (Air conditionné, essuies glass, verrouillage des portières, ...).

Bref, dans tous les PCBs de mon TFE, un module CAN est intégré. Il s'agit du MCP2562FD. À partir d'une communication UART de l'ESP32, le module CAN retranscrit les trame UART sous forme CAN tout le long du bus.

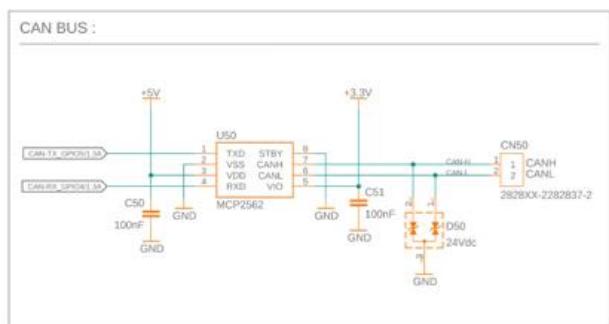


Figure 14 : CAN SCHEMA

Une résistance de 120Ω est placée de sorte à ce qu'elle court-circuite les lignes CAN L & CAN H. Il est possible de la débrancher depuis un jumper.

Dans le cas où le CANbus accueille plus de deux périphériques, alors seuls les périphériques positionnés aux extrémités du CANbus devront brancher le jumper (voir annexes 6, 12, 19 et 26).

3. Développement du sujet

3.1. Situation initiale du projet

Dans ce point 3.1, je développe mes idées et constats aux éventuels choix effectués lors de la situation initiale du projet. Je ne rentre pas dans les détails puisqu'il y aura un point adapté un peu plus loin dans ce dossier.

3.1.1. Développement des idées de la création des PCBs

Premières études :

Comme cité explicitement dans le cahier des charges, le projet doit embarquer plusieurs communications standardisées. Au départ, j'ai eu l'idée de concevoir un PCB équipant tous les modules de communications. Quant aux propositions du maître de stage, il avait plutôt en tête l'idée de concevoir plusieurs PCBs avec une communication embarquée propre au circuit imprimé lui-même.

Sans plus attendre, nous nous étions mis d'accord sur le fait de concevoir 4 PCBs. L'un venant de mon idée de départ, et les 3 autres venant de celles du maître de stage.

Modules de communications standardisées :

Le premier PCB a été le premier à voir ses idées & plans se développer. Il devait embarquer tous les modules de communications standardisées. Le choix des bons modules a scrupuleusement été étudié. La compatibilité électronique et la quantité des ressources disponibles sur le net de ces derniers étaient mes priorités pour effectuer un choix réfléchi et sans surprise lors de la mise en place de ces modules. Quant aux 3 autres PCBs, rien de sorcier, ils équiperont chacun d'entre eux un des 3 différents modules de communications (Zigbee, LoRa & SigFox) précédemment choisis.

Capteurs, actionneurs & alimentation :

Ensuite, cela m'a mené aux choix des composants autres que les modules radios : les capteurs et actionneurs. J'ai donc dressé une liste. Le défi était d'implémenter des capteurs & actionneurs différents de ceux du PCB développé par un ancien étudiant (actuellement utilisé dans le cours d'électronique) afin d'offrir une multitude de possibilités aux étudiants avec tous les PCBs réunis.

Le choix de la gestion d'alimentation a également été un sujet sensible. J'ai demandé à mon maître de stage quel type de circuit de puissance il souhaitait dans les PCBs. Il a précisé qu'ils n'intègreraient pas d'entrée extérieure par connecteur (9V par exemple), ni de sécurités embarquées, autre que celle micro USB 5V accessible depuis le microcontrôleur ESP32.

Les composants supportant une alimentation 5V, comme le ruban LED par exemple, seraient directement traités depuis la sortie 5V venant du micro USB. Quant aux composants ne supportant que 3.3V, la gestion se fera par le biais d'un régulateur de tension fixe 3.3V en partant du même 5V.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Supplément :

A ce stade, les idées étaient quasi finies et mises en place sur les plans. Pour une touche plus originale, il fallait ensuite faire place aux éventuels améliorations & modules supplémentaires. C'est là que l'idée d'ajout d'un keyboard et un afficheur 7 segments a été proposée par le maître de stage. Il s'est souvenu qu'un circuit intégré I²C en particulier pouvait gérer les deux en un : le TM1637.

Après plusieurs recherches sur ce nouveau composant, j'ai remarqué un détail qui nous a échappé lors du choix du TM1637. Ce IC n'est apte à ne lire que les keyboard 8x2 bit. Autrement dit, le keyboard que les futurs étudiants 3AU ont dans leur kit électronique ne sera pas compatible avec le TM1637 car il est de 4x4 bit. J'ai donc fait des recherches, et j'ai trouvé plusieurs IC similaires, mais ne supportant jamais les exigences demandées, excepté un module : le TM1650.

Ce TM1650 est le seul à avoir la capacité d'interpréter notre keypad du kit. Mais j'ai remarqué que son protocole de communication I²C n'est pas vraiment l'I²C qu'on connaît, mais plutôt un pseudo I²C... En d'autres termes, le constructeur a personnalisé son utilisation en le privant d'une adresse. Ce qui signifie que dans le bus I²C aucun autre périphérique ne pourra être branché (comme l'OLED & le MPU6050 qui étaient actuellement prévus dans le bus I²C), excepté lui-même.

De ce fait, sans perdre plus de temps, le maître de stage a rejeté l'idée d'ajouter un afficheur 7 segments, tout en gardant le keyboard, mais cette fois- ci avec un autre circuit intégré : le MCP23008. Avec cet IC nous étions certains que le keyboard pouvait fonctionner en I²C.

En deuxième temps, bien après la conception des premiers PCBs, j'ai été mené à réaliser une deuxième version du premier PCB. L'idée d'ajout d'un emplacement SPI pour un RFID a été proposée, aussi celle d'une extension I²C. De mon côté, j'ai proposé d'ajouter un module CAN dans ce même PCB.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.1.2. Périphériques de tous les PCBs**

Comme précisé dans le cahier des charges, le projet ne se basera pas qu'aux divers types de communications, mais également à la gestion de différents capteurs et actionneurs. Par conséquent, une étude sur le choix et la quantité de ces derniers a été mise en place pour offrir un maximum de solution lors de l'usage des PCB industrie 4.0. Etant donné que le TFE ne se limite pas à un PCB mais à 4 PCB, les périphériques qui sont cités ci-dessous correspondent à l'ensemble des capteurs situés dans tous les PCB :

Résistance variable :

Figure 15 : Potentiomètre

Comme périphérique utilisé en lecture ADC, j'ai choisi d'utiliser une résistance variable de 10kOhm. Il s'agit d'un potentiomètre verticale ajustable sur le dessus à simple tour avec un format à pattes traversantes.

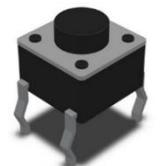
Boutons poussoirs :

Figure 16 : Boutons poussoirs

Comme entrées digitales, j'ai opéré à l'utilisation de deux boutons poussoirs. Couplés avec résistances de 10kOhm en mode pullup, les signaux booléens ignorent les interférences externes. Il s'agit de boutons bipolaires à pattes traversantes. Simple à utiliser, simple à programmer, ce périphérique est idéal pour chaque type d'application.

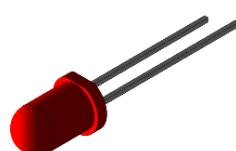


Figure 17 : LEDs

Des LEDs ont été utilisées en tant que sorties digitales. Elles sont placées en série avec une résistance 330Ohms pour limiter leurs courants à quelques dizaines de milliampères. Ayant un format de 5mm de diamètre, avec des pattes traversantes, ces périphériques informent à l'utilisateur l'état de son processus. Chacune d'entre elles n'ont pas toutes les mêmes couleurs : le rouge pour des signaux simple (LED1, LED2 & +3V3), le vert pour les signaux de communication UART, le bleu pour la communication radio et le jaune pour l'activation du CPU (celui du module Sigfox).

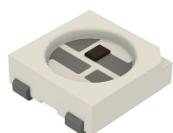
LEDs Neopixel :

Figure 18 : LEDs Neopixel

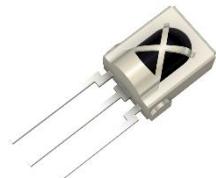
Comme autre type de sortie digitale, j'ai utilisé trois LEDs WS2812B-B. Ces derniers sont adressables à partir du protocole Neopixel émis par le microcontrôleur. Quel que soit le nombre de LED utilisés, l'ensemble de ces derniers sera dirigeable par une seule GPIO de l'ESP32. L'accès à des couleurs RGB est également possible, ainsi que le dimming.

LED infrarouge :

Figure 19 : LED infrarouge

En tant que nouvelle communication embarquée dans le PCB, j'ai choisi l'infrarouge. Cette LED fait office d'émettrice de signaux IR. Il s'agit de la LED L-7113F3BT équipée d'une longueur d'onde de 940nm. L'orientation de ses broches à 90° a été faite dans le but de favoriser le sens du signal photique.

Récepteur infrarouge :



Le TSOP18638 a été choisi comme récepteur de signaux infrarouges. Équipé d'une fréquence porteuse de 38kHz avec une distance de détection allant jusqu'à 24m, ce capteur est le composant idéal pour les différentes applications possibles à réaliser avec les PCB industrie 4.0.

Figure 20 : Récepteur infrarouge

Module PIR :



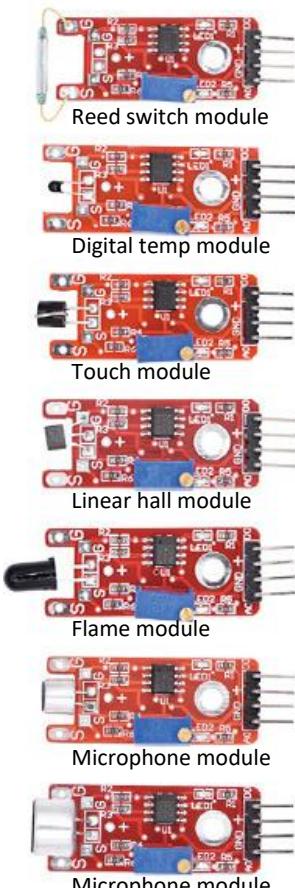
Figure 21 : Module PIR

En tant qu'autre périphérique d'entrée digitale, j'ai fait le choix d'utiliser un HC-SR501. Il s'alimente en 5V et envoie un signal de sortie en 3V3. Il s'agit d'un capteur pyroélectrique ou PIR. DéTECTANT un corps chaud en mouvement, ce composant est pour moi un capteur essentiel à comprendre et à dompter. La distance et le temps d'activation de sa sortie digitale est paramétrable à partir de deux potentiomètres situés en dessous de son PCB.

Le capteur analyse/surveille l'environnement de sa portée maximale. En partant du principe que chaque objet dégage sa propre onde infrarouge, si cet environnement vient à se modifier (un intrus entre dans l'environnement), le capteur s'enclenche.

Entre ce capteur et le microcontrôleur, aucun protocole n'a lieu. La technologie de détection est embarquée dans le capteur de façon à générer un niveau HAUT ou BAS en fonction de la détection.

Kit de capteurs d'Elegoo :



Un emplacement de capteurs de toute sorte est présent dans 3 des 4 PCB industrie 4.0. L'avantage de ces derniers est qu'ils intègrent tous une même carte mère, c'est-à-dire que les pinout sont identiques quel que soit le type de capteur.

Les signaux de ces modules sont tous sortants : A0 (sortie analogique) & D0 (sortie digitale). La sortie analogique est celle qui est amplifiée par l'amplificateur intégré dans la carte mère, et ce à partir d'un AOP. Il s'agit plus précisément d'un comparateur différentiel double (LM393). La sensibilité, ou plutôt le gain du circuit de la sortie A0 est réglable par le trimpot (résistance variable) située sur la carte mère elle-même.

Dans le cas où les capteurs sont de type tout ou rien, la sortie analogique est l'image de la sortie digitale.

Dans le kit acheté par les étudiants se trouvent donc tous ces capteurs. Parmi eux se trouvent :

- **Reed switch m.** : Contacts sensibles au magnétisme.
- **Digital temp m.** : Thermistance NTC (-55°C à 125°C).
- **Touch m.** : Broche flottante d'un transistor Darlington
- **Linear hall m.** : Capteur à effet Hall linéaire 49E.
- **Flame m.** : Récepteur IR dimensionné aux infrarouges émises par le feu.
- **Microphone m.** : Grand petit microphone à condensateur électrique.
- **ECM Sensor m.** : Grand microphone à condensateur électrique.

Figure 22 : Kit de capteurs d'Elegoo

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

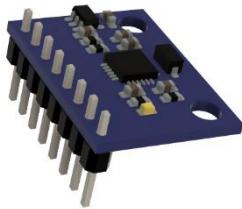
Accéléromètre & gyroscope :

Figure 23 : MPU 6050

Le MPU 6050 est un capteur permettant de mesurer des accélérations linéaires (accéléromètre) & les vitesses angulaires (gyroscope). Alimenté en 3.3V, le protocole de communication entre le capteur et le microcontrôleur est l'I²C.

Pour faire simple, accéléromètre permet de récupérer en trois dimensions, x y et z, les inclinaisons et vibrations d'un objet.

L'accéléromètre est utilisé dans les smartphones, les tablettes, et autres appareils courants. Celui-ci est utilisé par exemple dans les réalité virtuelle, tel que le jeu « Pokémon Go ». En effet, ce jeu vidéo reprend les données des trois dimensions afin de positionner synchroniquement un Pokémon virtuel dans un environnement figuré par la caméra du GSM.

Le MPU 6050 inclut également un gyroscope. Ce dernier permet de mesurer/récupérer les vitesses angulaires d'un objet exprimés en degré/s. Il récupère donc les changements d'angles effectués par unité de seconde.

Nous trouvons ces deux fonctions du capteur dans les drones. Cela permet de les stabiliser au mieux dans l'air. En effet, lorsqu'un utilisateur utilise une télécommande d'un drone, il ne gère pas les variations ou la stabilité de l'objet lui-même, mais plutôt les sens et directions dans lequel il faut diriger le véhicule.

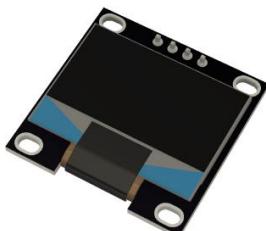
Ecran OLED :

Figure 24 : OLED

Ce périphérique utilise le protocole de communication identique au MPU6050, l'I²C. Il s'agit d'un écran SSD1306, de 0.96 pouce (128 x 64 pixels) permettant d'afficher des données acquises voire même des images. Il est alimenté en 3.3V.

Chaque pixel étant une LED, cette technologie offre une large gamme d'application. Aujourd'hui, on en retrouve dans certains smartphones, écrans de télévision etc... Plus compact, plus facile à construire, plus rapide en termes de rafraîchissement, cette technologie remplace petit à petit les afficheurs LCD.

Keyboard & I/O expander I²C :

Figure 25 : MCP 23008

Ce circuit intégré propose au PCB une extension de 8 pins digitales (entrées/sorties). Alimenté en 3.3V, il s'agit d'un MCP23008. A nouveau, ce périphérique est adressé en I²C.

Il est possible d'avoir dans un même bus I²C jusqu'à 8x un MCP23008. Le changement d'adresse I²C s'effectue par les broches A0 A1 & A2 en forçant leur niveau haut ou bas ($2^3 = 8$).

Ce circuit intégré initialement choisi pour la gestion d'un keyboard, il a la capacité d'exercer un tas d'autres tâches étant donné qu'il dispose d'entrées/sorties quasiment classiques, tel que la gestion d'un afficheur 7 segments, de boutons poussoirs, de LEDs, d'éventuels capteurs...



Figure 26 : Keyboard

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Photorésistance LDR :

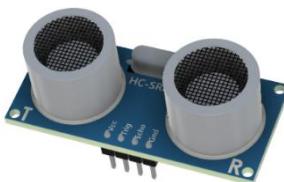
En tant qu'entrée ADC, le niveau de luminosité ambiante est captable grâce à une LDR. Le signal d'entrée peut varier entre 0 et 3.3V par le biais d'un pont diviseur de tension. Ce pont diviseur est alimenté par une source de tension 3.3V.

Figure 27 : LDR

Capteur de température numérique :

Le DS18B20 est un capteur de température numérique. Il est lié à une entrée digitale du microcontrôleur. Sa plage de détection de température peut aller de -55 à 125°C. Il est doté d'une précision de +/- 0.5°C. Il est doté d'une résolution allant jusqu'à 9 bits. Etant donné qu'il s'agit d'un capteur numérique, l'intervalle d'acquisition de température est limité à toutes les 750ms.

Ce capteur utilise le protocole 1-Wire pour communiquer avec le microcontrôleur. Ainsi, grâce à une GPIO du microcontrôleur, il est possible de brancher plusieurs capteurs du même genre.

Capteur ultrasonique :

Il s'agit du modèle HC-SR04. Ce périphérique occupe deux broches digitales. L'une étant une entrée et l'autre une sortie. Il est alimenté en 5V et renvoie le signal de détection en 5V également, ce qui pose un problème pour le microcontrôleur. Un pont diviseur de tension a été indispensable lors de son utilisation parce que le microcontrôleur intègre une tension logique de 3.3V.

Figure 29 : HC-SR04

Un capteur ultrasonique envoie constamment des impulsions sonores à haute fréquence. Ces signaux se déplacent à la vitesse du son. Ce capteur détecte des obstacles avec les "échos" qu'il reçoit en retour. S'il ne détecte rien, c'est qu'il n'a pas de retour, soit aucun obstacle. Le capteur tiendra compte d'un facteur de l'écho afin d'envoyer un signal à son automate : le temps de retour. Celui-ci sera l'image de la distance. Cette dernière se calcule dans le programme encodé. Il calcule la distance le séparant de l'obstacle sur base du temps écoulé entre l'émission du signal et la réception de l'écho.

Buzzer :

Figure 30 : Buzzer

Ce buzzer occupe une seule GPIO digitale. De type piézoélectrique, ce composant est utilisé comme signalisation audio. A partir d'un signal fréquentiel, ce dernier peut émettre des sons. Plus la fréquence est grande, plus le son devient aigu.

Il n'est composé que de deux pattes, l'une étant (négative) le GND, l'autre (positive) l'alimentation/signal. Ses signaux de commande peuvent aller jusqu'à 5V. Etant donné que le microcontrôleur a une tension logique de 3.3V, la solution a été de passer par un circuit transistorisé depuis un mosfet.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Servo moteur :

Figure 31 : Servomoteur

Ce servomoteur occupe une seule sortie digitale du microcontrôleur. Il est équipé de 3 broches, l'une étant (l'alimentation) 5V, l'autre GND et la dernière le signal.

Un servomoteur est équipé d'un moteur DC, d'un réducteur mécanique (permettant d'augmenter le couple de ce dernier), d'un potentiomètre (lié physiquement à l'axe de rotation du servomoteur) et d'une carte électronique (simulant le principe de driver, simulant le principe d'une régulation sur base d'une consigne, et interpréter les signaux du microcontrôleur).

Le message envoyé depuis le microcontrôleur est transporté par une porteuse, une fréquence ayant une période variable proportionnelle à l'angle de rotation de l'axe souhaité.

En fonction de la période du signal, l'électronique embarquée dans le servomoteur devinera l'angle souhaité correspondant. Le moteur DC se mettra à tourner. Il commencera à s'arrêter, ou à changer de sens lorsque le potentiomètre indique que la position est atteinte. Le moteur s'adaptera à l'angle lu par le potentiomètre de façon à constamment respecter la consigne (angle).

DHT22 :

Figure 32 : DHT22

Le DHT22 est un capteur de température et d'humidité. Il n'occupe qu'une seule GPIO digitale du microcontrôleur. Identiquement au capteur DS18B20, celui-ci est également un capteur numérique. Sa plage de détection de température peut aller de -40 à 125°C avec une précision de +/- 0.5°C. Sa plage de détection d'humidité peut aller de 0 à 100% avec une précision de +/- 2.5%. Étant donné qu'il s'agit d'un capteur numérique, l'intervalle d'acquisition de ces données utiles sont limités à toutes les 500ms.



Figure 33 : RFID

Ce détecteur est branché depuis un bus SPI (MOSI, MISO, CLK). De plus, il occupe deux GPIO digitales du microcontrôleur (CS & RST). Le principe du bus SPI sera expliqué dans les pages qui suivent.

Ce composant détecte la présence d'une carte RFID présente à sa portée. Il récupère ensuite les identifiants uniques de cette carte (équipée d'une puce).

Le principe de l'RFID (Radio Frequency Identification) utilise des signaux radio pour communiquer à courte distance. Le système proposé par ce détecteur est le suivant :

Son antenne envoie constamment un signal radio. Lorsque la puce de la carte d'identification RFID est à la portée du signal radio, elle s'alimente grâce à ce signal reçu. Une fois alimentée, cette puce émet des signaux d'identification vers le détecteur.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**Emplacement MikroBus :**

Figure 34 : Click Boards of MikroBUS

Un emplacement MikroBus dans un PCB permet d'accueillir des click boards (présenté en vert ci-contre). On retrouve plusieurs milliers de click boards dans le marché. Une fois insérés dans le PCB, ces derniers peuvent apporter une infinité de possibilités, et le tout disponible à partir d'un minimum de broches.

Des click boards, on en trouve de toute sorte, des capteurs, des modules radio, des écrans, etc...

Leur polyvalence apporte non seulement une infinité de solutions à notre PCB, mais aussi une simplicité au niveau de leur gestion. Il est possible de communiquer avec eux depuis un bus I²C, SPI, UART, ...

Voici le footprint (empreinte) que propose mikroBUS pour accueillir n'importe quel click boards :

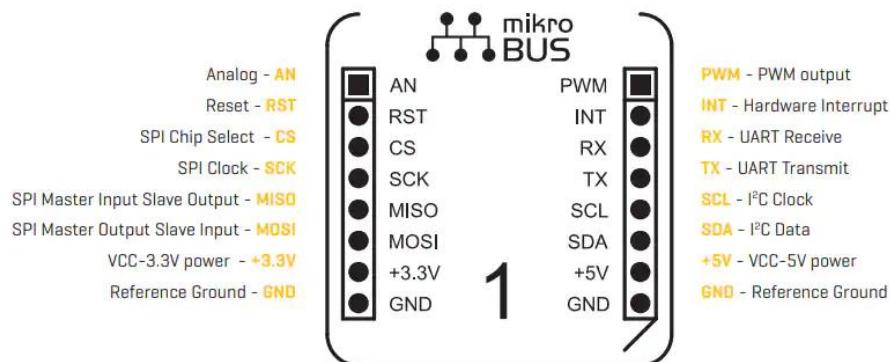


Figure 35 : MikroBUS Pinouts

3.1.3. Analyse des solutions d'un TP Industrie 4.0

Premières études : TP Zigbee2MQTT

Le maître de stage était arrivé en classe avec un dongle particulièrement intéressant et m'a proposé de le faire fonctionner et d'en créer une application. Il s'agissait d'un dongle me permettant d'interpréter les trames Zigbee (préalablement envoyées par les modules Zigbee de plusieurs différents PCBs) et de les publier en MQTT.

En d'autres termes, l'idée serait qu'en insérant ce dongle dans un PC, on serait capable d'observer et de lire tout le maillage domotique effectué par le biais de chaque PCB communiquant entre eux en Zigbee. Ce dongle s'identifie donc à une passerelle entre deux différents réseaux du moins très intéressants.

J'ai précédemment parlé de MQTT. Il s'agit d'un protocole de messagerie basé sur le protocole TCP/IP qui peut avoir lieu dans la bande passante du Wi-Fi ou dans les câbles RJ45. Idéal dans des installations domestiques et/ou industrielles.

Cette couche MQTT doit être gérée par un serveur, tel un PC ou un Raspberry Pi. Pour des raisons matérielles, prochainement expliquées, j'ai fait le choix d'utiliser un RPI4 à la place d'un PC. De plus, il était nécessaire d'installer le plugin Zigbee2MQTT en tant que conteneur Docker pour me permettre de configurer & lire en UART le dongle depuis l'USB du Raspberry Pi 4. Les données utiles lues en UART sont redirigées en MQTT. Comme deuxième conteneur, il y avait justement celui de la création du MQTT.

Les études que j'ai menées dessus ont pris beaucoup de temps. J'ai réussi à observer le maillage de mes différents PCBs connectés par le biais du dongle. J'étais arrivé à un stade où je détectais leurs statuts, leurs firmware, etc... mais pas encore leurs données utiles (éventuellement la température, l'humidité, l'appui sur un bouton, ...). Après quelques autres moments de recherches, je me suis rendu compte que le conteneur Zigbee2MQTT ne supportait pas entièrement les modules Zigbee qui sont embarqués dans les PCBs. En conclusion, je ne pouvais plus aller plus loin dans cette application munie de la passerelle Zigbee2MQTT.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**Deuxième étude : TP Multi-Protocole**

C'est alors que je me suis directement mis à réaliser un autre type d'application similaire sans perdre plus de temps. Je n'ai pas eu de mauvaises surprises cette fois- ci. Voici un schéma représentant ce nouveau TP :

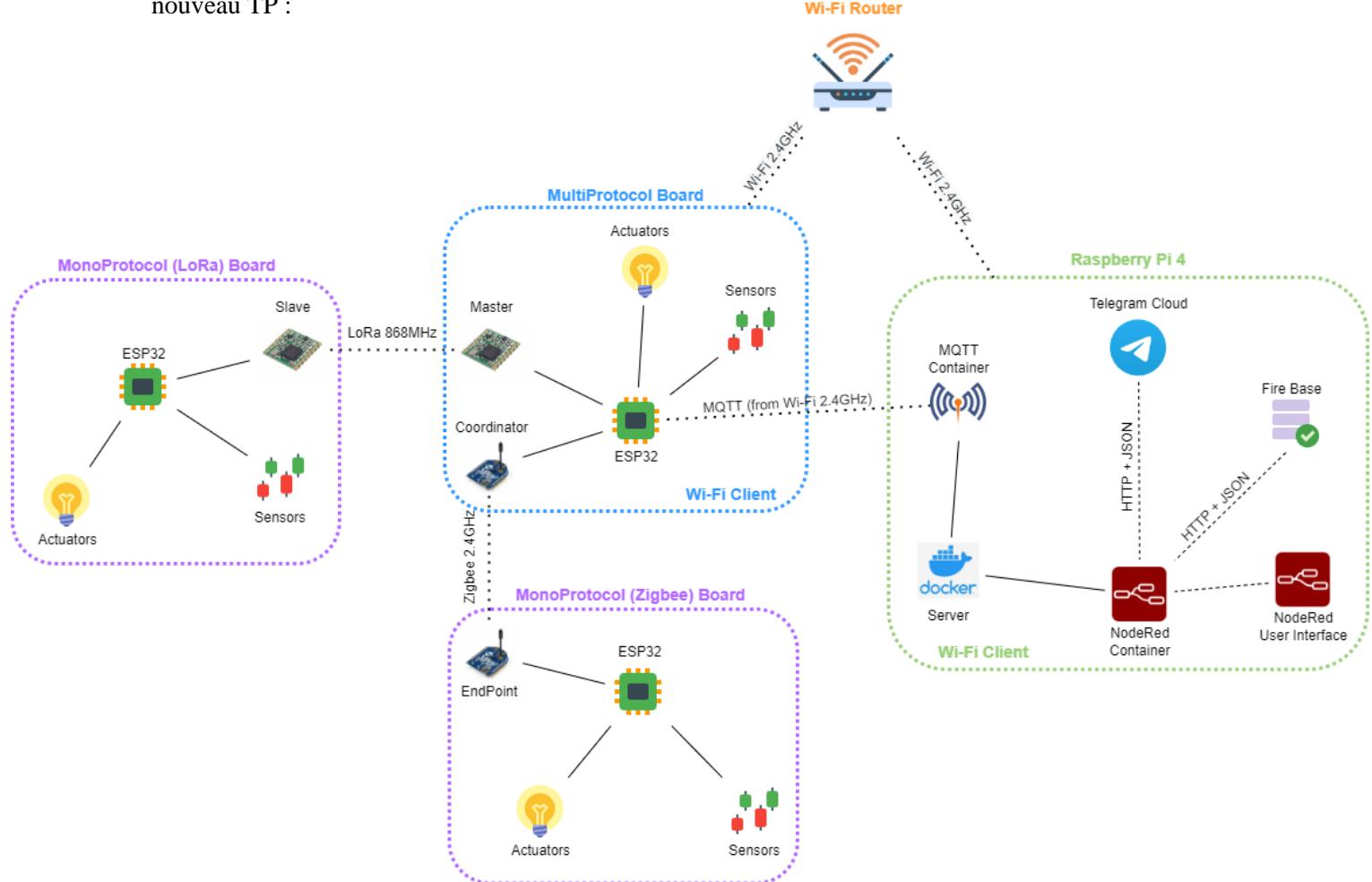


Figure 36 : TP Industry 4.0 - TP Multi-Protocol

Toujours avec un Raspberry Pi 4, l'idée de ce TP serait d'assembler 3 des 4 PCBs et de les faire communiquer entre eux, mais pas n'importe comment. En premier temps, le **PCB multi-protocole** et le **Raspberry Pi 4** doivent se connecter en tant que client Wi-Fi à partir d'un point d'accès. Ces deux derniers communiqueront entre eux sous la couche MQTT créée par le **Raspberry Pi 4**. En deuxième temps, le **PCB orienté Zigbee** communique en radio avec le **PCB multi-protocole** et le **PCB orienté LoRa** communique à son tour également en radio avec le **PCB multi-protocole** indépendamment.

Le **Raspberry Pi 4** simulera un bot depuis le cloud Telegram. Sur base des interactions effectuées dans le tchat entre l'utilisateur et le bot, le **Raspberry Pi 4** demandera au **PCB multi-protocole** en MQTT d'agir en conséquence.

Par exemple, si l'utilisateur demande d'allumer la LED 2 du **PCB orienté LoRa**, alors le **Raspberry Pi 4** envoie la requête correspondante en MQTT aux **PCB multi-protocole** pour qu'à son tour ce dernier envoie la même requête en radio sous le protocole LoRa vers le **PCB orienté LoRa**. Il allumera sa LED et enverra un accusé de réception (en renvoyant une requête en sens inverse) de sorte que l'utilisateur sache que sa demande ait bien été exécutée.

Toutes les requêtes ainsi que leurs réponses sont enregistrées dans une base de données (Fire Base). Leur visualisation peut se faire depuis l'interaction dans le tchat entre l'utilisateur et le bot.

3.2. Mise en œuvre de la solution hardware

3.2.1. PCB Multi-Protocole

Le PCB multi-protocole est polyvalent. Il embarque en lui la majorité des périphériques qu'ont les 3 autres PCBs. Les modules de communication standardisés embarqués sont ceux des protocoles LoRa, CAN, SigFox, Zigbee, & infrarouge. On y retrouve également ceux du Wi-Fi et du bluetooth parce qu'ils sont nativement embarqués depuis le microcontrôleur ESP32.

Ce dernier gère tous les périphériques. Certains d'entre eux sont adressables par le biais de protocoles pré-disponibles depuis l'ESP32. Je fais référence au bus I²C qui gère l'OLED, le MPU6050 & le MCP23008. Il y a aussi le bus SPI qui, à son tour, gère le RFID et le module LoRa. On retrouve aussi le protocole UART qui dirige les modules CAN, SigFox & Zigbee.

L'ESP32 embarque aussi l'interface analogique. Il convertit une donnée analogique en une donnée numérique, interprétable par le microprocesseur du microcontrôleur lui-même. Ce dernier reprend le principe d'échantillonnage. Proportionnellement à une tension analogique en entrée du GPIO allant de 0 à 3.3V, le micro-processeur voit un niveau numérique entre 0 et 4095. Il s'agit d'une précision de 12 bits. Les périphériques ayant ce type de sortie analogique sont le potentiomètre et le capteur d'Elegoo.

Evidemment, l'esp32 gère des périphériques de type tout ou rien, par le biais de ses GPIO digitales. Les périphériques concernés sont le capteur de présence PIR, le capteur d'Elegoo et les LEDs et les boutons.

A partir de broche digitale, il est possible de créer des protocoles autres que ceux présentés ci-dessus. Grace aux bibliothèques que nous pouvons inclure dans le programme du microcontrôleur, il est donc possible de simuler des protocoles. Il y a celui du Neopixel par exemple, il peut gérer une infinité de LEDs RGB adressable (un ruban LED) à partir d'une seule GPIO. Il y a également le protocole propre aux capteurs DHT22, nous permettant de récupérer la température et l'humidité le tout en un fil de donnée. Il y a aussi l'infrarouge, simulant le principe de télécommande et de récepteur d'une télévision par exemple.

Voici un bref résumé des périphériques qu'inclue le PCB Multi-Protocole sous forme de tableau :

Analogique	Digitale	SPI	UART	I ² C	Autre
Potentiomètre	PIR	RFID	CAN	OLED	LEDs Neopixel
Capteur Elegoo A0 & D0	LoRa	SigFox	MPU6050	DHT22	Infrarouge
LED 1&2		Zigbee	MCP2300		
BP 1&2					

3.2.1.1. Schema bloc



Figure 37 : Schema bloc - PCB Multi-Protocole

L'idée d'un schéma bloc est celle d'avoir un support visuel dont nous pouvons nous baser pour avoir une image simplifiée de l'architecture des systèmes du PCB.

L'image parle donc d'elle-même. A partir de plusieurs différentes interfaces de communication (Digitale, SPI, UART, I²C, Analogique, etc...), l'ESP32 gère tous les périphériques présents sur le PCB.

3.2.1.2. ESP32 Pinout

Multi Protocol V2		
GPIO	Devices	Notes
0	LORA-NSS	SPI Chip Select
1	/	
2	LORA-DIO0	
3	/	
4	IO4 Jumper	Jumper JP11: Choice of CAN-RX or LED1
5	IO5 Jumper	Jumper JP12: Choice of CAN-TX or BP2 (HIGH when pushed)
12	LED2+RFID-NSS	LED ON when LOW+SPI Chip Select
13	PIR	HIGH when detected
14	LORA-RST	
15	IO15 Jumper	Jumper JP14: Choice of RFID-RST or Neopixel LEDs
16	XBEE-TX	UART2
17	XBEE-RX	UART2
18	RFID+LORA-SCK	SPI Clock
19	RFID+LORA-MISO	SPI MISO
21	SDA (I ² C)	Devices: OLED; MCP23008; MPU6050
22	SCL (I ² C)	
23	RFID+LORA-MOSI	SPI MOSI
25	Sigfox-TX	SoftwareSerial
26	SigFox-RX	SoftwareSerial
27	DHT22	
32	IR-Sender	
33	IR-Receiver	
34	E.Sensors-A0	
35	RV	
36	BP1	HIGH when pushed
39	E.Sensors-D0	

Ce tableau représente les liaisons hardware entre les pins de l'ESP32 et des différents périphériques. Sous forme de schéma électronique, on peut le retrouver en annexe en fin du dossier (voir annexes 2 à 6).

La couleur de police en jaune représente le fait que les GPIO de l'ESP32 concernés sont pour des utilisations d'interfaçage SPI avec son bus. La couleur du surlignage des textes en jaune correspond à des emplacements de jumpers liés aux GPIO de l'ESP32. Et du côté de la couleur du surlignage des textes en bleu, cela signifie qu'il s'agit de GPIO gérant un bus I²C.

Certains pins ne doivent pas être utilisés n'importe comment. Lorsque la broche GPIO 0 de l'ESP32 est utilisée, plusieurs exigences doivent être respectées. A condition qu'aucune résistance de type pull up/down ne soit branchée, et que son utilisation ne soit pas de type PWM, nous pouvons y insérer une entrée ou sortie. Dans mon cas, je l'ai branché au chip select (expliqué plus loin lors de l'étude du protocole SPI du module LoRa).

Je parcourrai d'autres conditions aux pages suivantes étant donné que le TFE se compose de 4 PCB.

3.2.1.3. Schémas électroniques

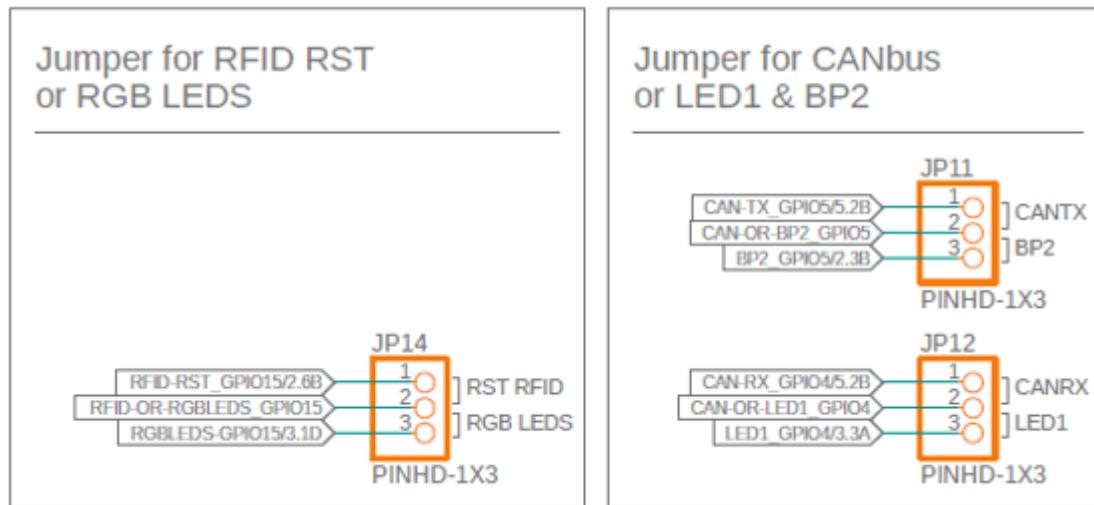


Figure 38 : SCHEMA Jumpers - PCB Multi-Protocole

Les pins GPIO4 & 5 ont également une particularité. Ce sont les seuls à simuler de l'UART pour une utilisation CAN Bus. D'ailleurs, dans le cas de mon application, j'utilise ces GPIO pour deux autres périphériques également, en plus du module CAN. Il s'agit de la LED1 et du BP2. Le choix entre l'utilisation de ces derniers ou du CAN se fait par le biais de jumper justement (voir annexe 2).

On peut remarquer que j'ai également utilisé un autre jumper au GPIO15. Il permet l'utilisation du RFID ou des LEDs Neopixel.

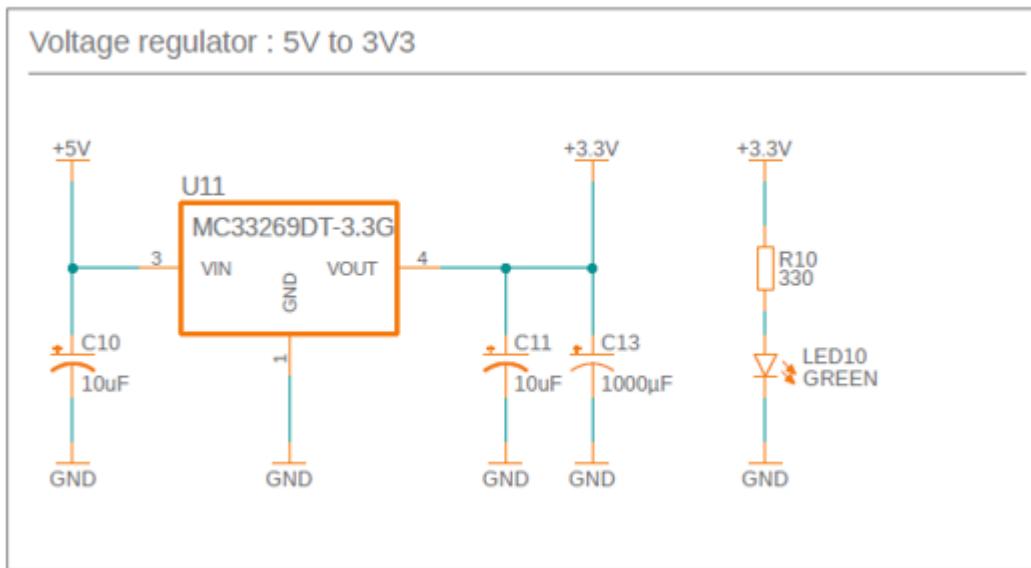


Figure 39 : SCHEMA Alimentation - PCB Multi-Protocole

Le choix de l'alimentation se fait par le biais d'un régulateur de tension fixe. Il s'agit du mc33269dt-3.3g. Il s'agit du même régulateur qu'équipe un PCB de prototypage que nous avons utilisés dans le cours de systèmes embarqués : l'EasyPIC.

Ce régulateur a besoin de condensateur en entrée et en sortie pour qu'il puisse générer un 3V3 depuis le 5V dans de bonnes conditions. Son Datasheet conseille une valeur 10 μ F de part et d'autre. Il s'agit de condensateur de découplage.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

En général, depuis une piste commune en sortie du régulateur, on relie tous les périphériques d'un PCB. Les changements de courants produits par ces périphériques peuvent créer des chutes de tension non négligeables qui pourraient affecter le bon fonctionnement des éléments voisins (circuits intégrés, transistors, ...) connectés à cette piste commune. Si c'est le cas, c'est à cet instant que les condensateurs de découplage entrent en jeu. En effet, au lieu de provoquer une instabilité sur la piste commune, ces condensateurs dévieront les courants de sorte que les chutes de tension soient amorties sur toute la piste commune, n'influençant ainsi aucun élément voisin, dont la source : le régulateur de tension. Ces condensateurs augmentent donc considérablement la durée de vie des composants électroniques.

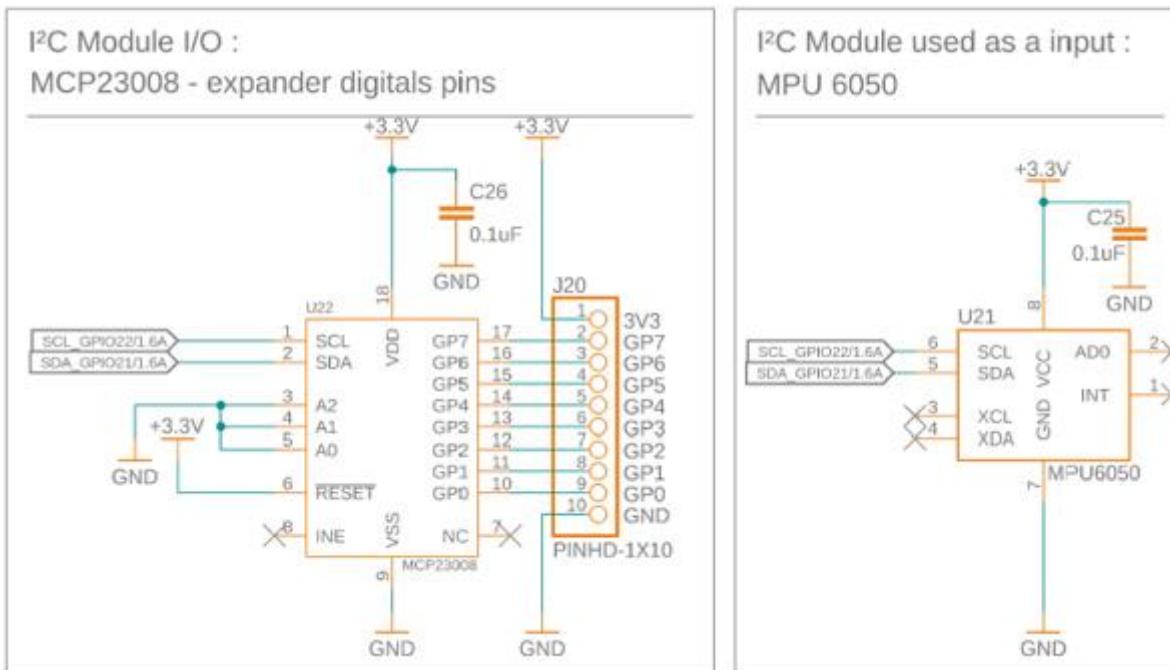


Figure 40 : SCHEMA I²C modules - PCB Multi-Protocole

Comme on peut le voir ci-dessus (voir annexe 3), on en retrouve aussi du côté des périphériques (C25, C26). Dans cet exemple, il n'y a que les modules MCP23008 & MPU6050 qui sont affichés, mais rassurez-vous, tous les autres périphériques du PCB équipent aussi un de ces condensateurs près de leur broche d'alimentation.

Il y a plusieurs autres aspects que j'aimerais expliquer dans les schémas de principes de ce PCB Multi-Protocole, mais ils sont répétés dans les 3 PCBs suivants. Je passerai donc en revue l'entièreté de l'architecture des schémas électriques au fur et à mesure en parcourant tous les PCBs dans les pages qui suivent pour éviter de me répéter à chaque fois.

3.2.1.4. Rendu final



Figure 41 : 3D - PCB Multi-Protocole

Le rendu final n'a pas pu être aussi attristant sans l'ajout d'images & logos sur les PCBs (voir annexe 8). Pour ce faire j'ai fait appel à quelques étapes de traitements d'images avant d'en arriver à ce résultat :

En premier temps, je cherche une image qui correspond à mes attentes sur un navigateur. Puis, en le copiant et collant sur le programme Inkscape, je le vectorise :

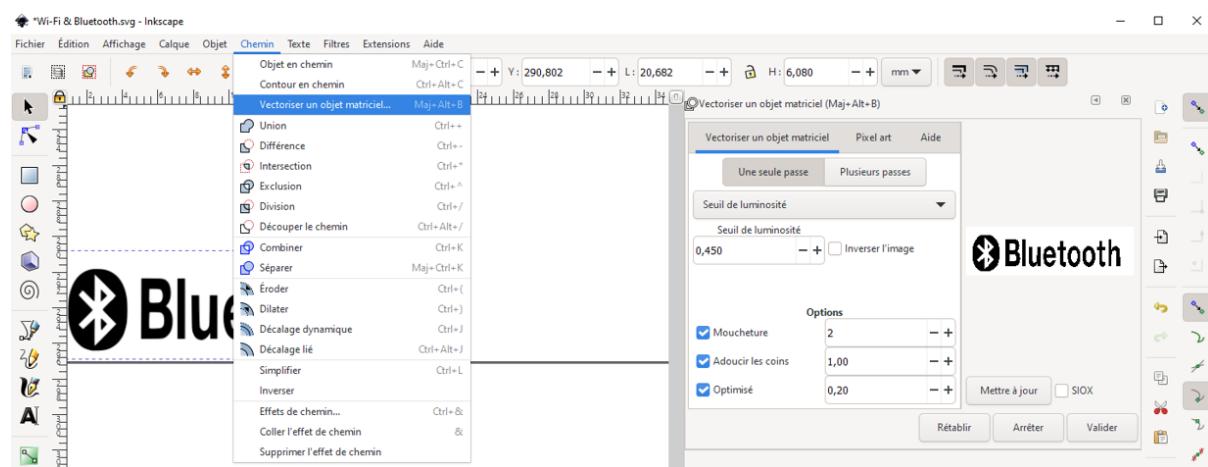


Figure 42 : Inkscape - PCB Multi-Protocole

Une fois que l'image a été vectorisée, j'enregistre le projet sous l'extension SVG. Ce dernier doit ensuite être importé dans un site permettant de convertir le SVG en Script Eagle : <https://gwilliams.github.io/svgtoeagle/>.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Une fois que le script est ajouté dans notre presse papier, il m'a suffi de le copier dans la ligne de commande de Fusion360 dans le projet PCB :

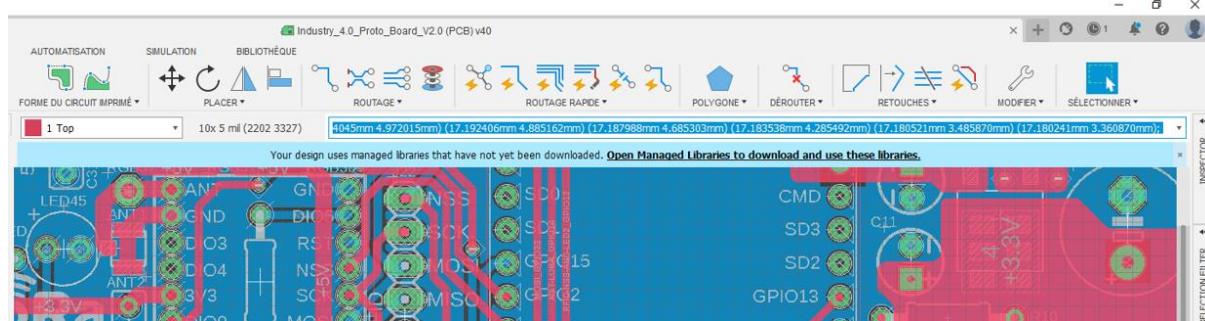


Figure 43 : Ligne de commande Fusion360 - PCB Multi-Protocole

En exécutant cette ligne de commande, l'image apparaitra aux positions/coordonnées précédemment choisies depuis Inkscape. Il suffira de déplacer l'image sur Fusion360 si sa position est décalée. Voici le résultat :

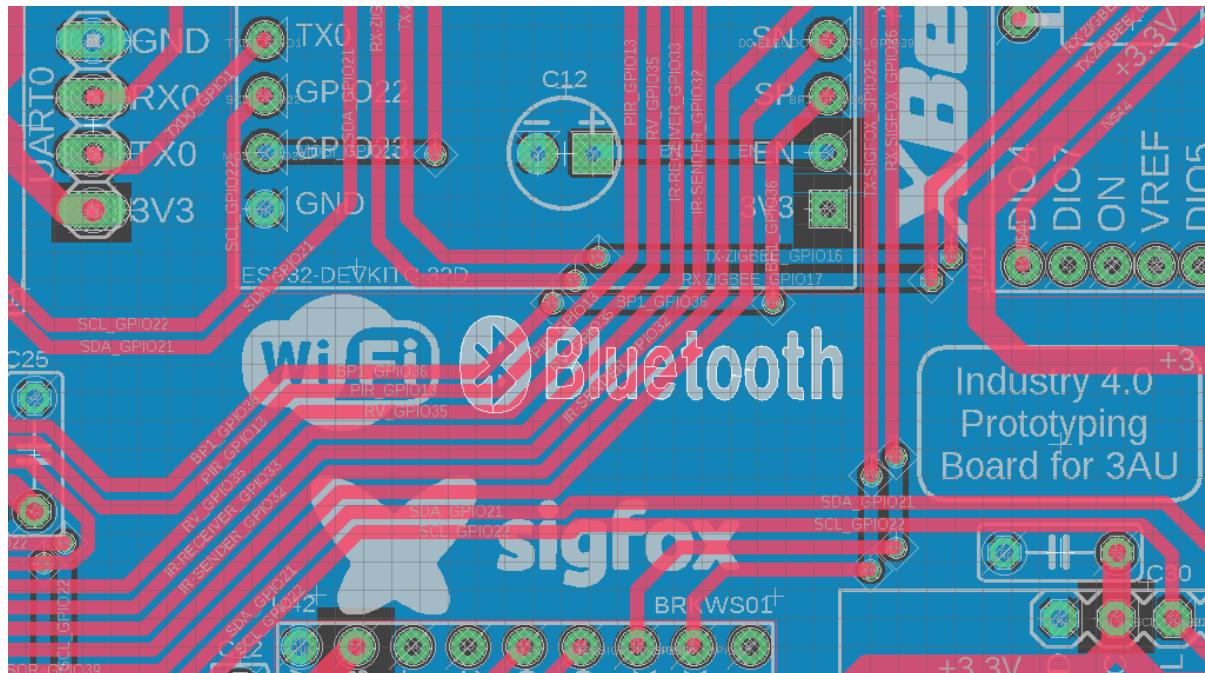


Figure 44 : Résultat SVG to PCB - PCB Multi-Protocole

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.2.2. PCB Mono-Protocole orienté Zigbee**

Le PCB mono-protocole orienté Zigbee est moins polyvalent que le précédent (PCB multi-protocole). Il mesure 85mm x 95mm. Il n'embarque pas exactement les mêmes périphériques. En effet, les modules de communication standardisés qu'il embarque sont ceux des protocoles CAN, I²C, Wi-Fi, Bluetooth et enfin, le Zigbee évidemment.

Le bus I²C gère l'OLED, & le MCP23008. Grâce à une extension physique de l'interface I²C, il est aussi possible d'ajouter d'autres périphériques dans ce bus, par le biais d'un connecteur femelle. On retrouve aussi le protocole UART qui dirige les modules CAN et Zigbee.

Quant aux périphériques de type analogique, on ne retrouve que le capteur d'Elegoo.

Du coté des périphériques dit digitales, le capteur d'Elegoo est également présent, les LEDs et les boutons poussoirs pareillement, mais aussi le buzzer. En effet, ce dernier peut être utilisé autant numériquement que de façon fréquentielle.

Et enfin, en simulant des protocoles depuis les broches digitales, on retrouve le protocole Néopixel, permettant la gestion de LEDs RGB adressables (un ruban LED) à partir d'une seule GPIO. On retrouve également un capteur capacitif, ou plutôt un Touch Button.

La gestion de ce capteur ne suit aucune interface précédemment expliquée (PWM, Analogique, Digitale, ...). Il faut savoir que l'esp32 est équipé de 10 GPIO embarquant nativement cette fonctionnalité capacitif. Toute variation de charge électrique peut être détectée depuis ces pins. La peau humaine peut stocker des charges électriques par exemple.

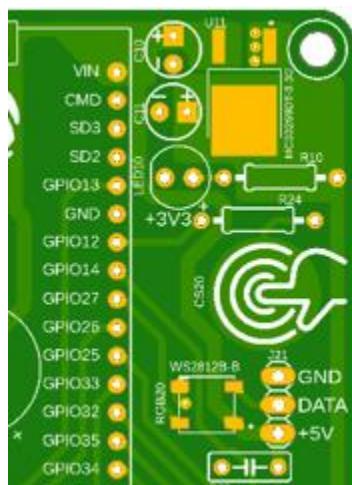


Figure 45 : PCB touch button - PCB Mono-Protocole Zigbee

L'électrode que j'ai placé dans le PCB (représenté sur l'image par un doigt) est une surface cuivrée prête à accueillir toute variation de charge électrique (voir annexe 15).

En fonction de la quantité de charge électrique que possède un individu, la valeur interprétée par l'ESP32 sera plus ou moins grande.

Rappelons le fait qu'un condensateur est équipé de deux armatures parallèles isolées par un diélectrique. La valeur de la capacité varie principalement en fonction de la distance isolante (épaisseur du diélectrique), et de la surface des armatures conductives.

De ce fait, si nous essayons de modifier un de ces paramètres, la capacité (en Farad) se voit modifier en conséquence.

En ajoutant ainsi en parallèle un autre condensateur (C_F = doigt humain), la surface totale des armatures se voit doubler, l'équivalent d'une capacité doublée ($C_{TOT} = C_F + C_P$).

La charge électrique que possédait cet autre condensateur est détectée par le circuit, en d'autres termes, par l'ESP32.

Cependant, des interférences externes peuvent influencer le circuit. L'ESP32 a prévu le coup en ajoutant plusieurs autres condensateurs.

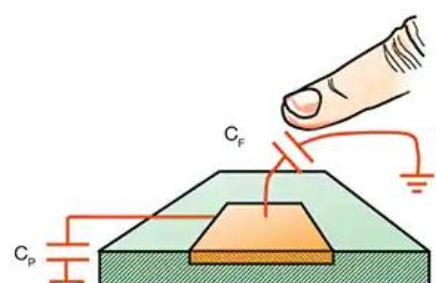


Figure 46 : Touch Button processus

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Bref, voici un résumé des périphériques qu'inclut le PCB mono-protocole orienté Zigbee sous forme de tableau suivit du schéma bloc :

Analogique	Digitale	SPI	UART	I²C	Autre
Mikrobus	Mikrobus	Mikrobus	Mikrobus	Mikrobus	Mikrobus
Capteur Elegoo A0 & D0	BP1&2		CAB	OLED	Touch Button
	LED1&2 Buzzer		XBee	MCP23008	LED Neopixel

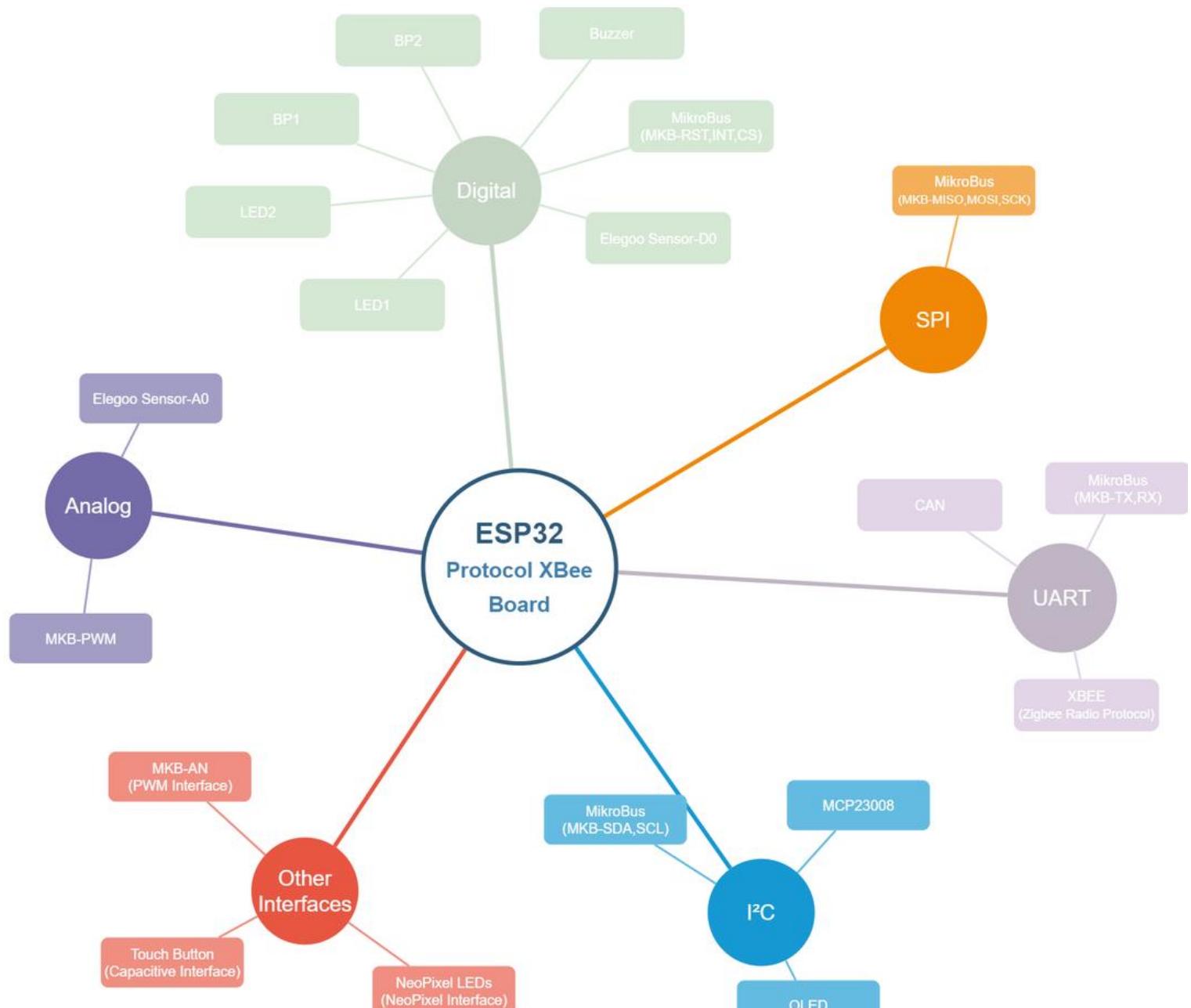
3.2.2.1. Schema bloc

Figure 47 : Schema bloc - PCB Mono-Protocole Zigbee

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

3.2.2.2. ESP32 Pinout

ESP32	Protocol Xbee V1.1	
GPIO	Devices	Notes
0	MPC23008-INT	
1	/	
2	Buzzer	
3	/	
4	CAN-RX	
5	CAN-TX	
12	Touch Button	
13	MKB-RST	MikroBus
14	LED1	LED active when HIGH
15	MKB-INT	MikroBus
16	IO16 Jumper	Jumper JP12 : Choice of MKB-TX or XBEE-RX; UART2
17	IO17 Jumper	Jumper JP13 : Choice of MKB-RX or XBEE-TX; UART2
18	MKB-SCK	MikroBus+SPI Clock
19	MKB-MISO	MikroBus+SPI MISO
21	SDA (I²C)	Devices: OLED; MCP23008; MKB-SDA,SCL(MikroBus)
22	SCL (I²C)	
23	MKB-MOSI	MikroBus+SPI MOSI
25	E.Sensors-D0	
26	MKB-PWM	MikroBus
27	Neopixel	
32	LED2	LED ON when HIGH
33	MKB-CS	MikroBus+SPI Chip Select
34	BP1	HIGH when pushed
35	BP2	HIGH when pushed
36	MKB-AN	MikroBus
39	E.Sensors-A0	

Ce PCB Mono-Protocole orienté Zigbee est équipé de deux jumpers. Ils permettent de mapper l'utilisation de l'UART2 vers les modules XBee, ou mikrobus (voir annexe 9).

En vert, dans le tableau, on peut remarquer que le mikrobus est présent dans la majorité des GPIO de l'ESP32. L'avantage de l'avoir placé dans le PCB est celui de pouvoir exploiter les fonctionnalités d'une infinité de module mikrobus.

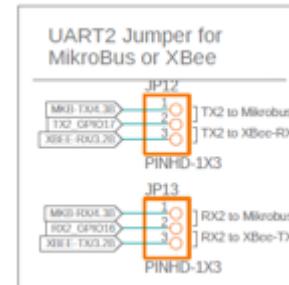


Figure 48 : SCHEMA jumpers - PCB Mono-Protocole Zigbee

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.2.2.3. Schémas électroniques**

En parlant de mikrobus, voici son branchement :

Il est branché à beaucoup d'interfaces de communication avec l'ESP32 (voir annexe 12). Les Click Boards que nous plaçons dans cet emplacement mikrobus n'exploitent jamais toutes ces interfaces. Certains n'utilisent que l'I²C, d'autres l'UART etc...

Leurs alimentations peuvent varier selon les applications du Click Board, certains acceptent le 3.3V, d'autres le 5V.

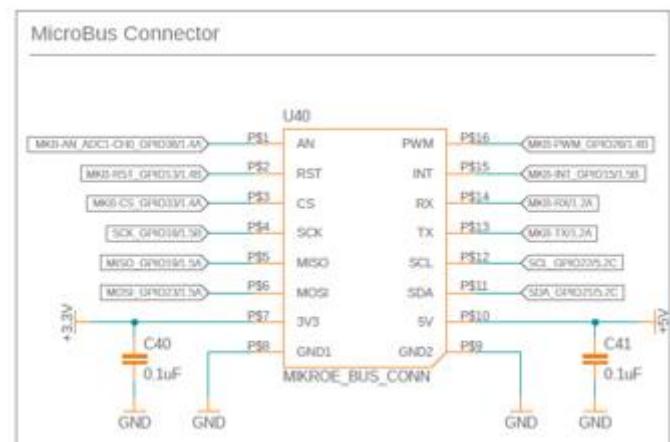


Figure 49 : SCHEMA MikroBUS - PCB Mono-Protocole Zigbee

Ce PCB orienté Zigbee a un autre périphérique qui n'est pas présent dans le PCB Multi-Protocole en plus du Touch Button : le Buzzer (voir annexe 10).

Un transistor mosphet BS170 est placé pour qu'il s'occupe d'isoler la puissance (Buzzer) de la commande (ESP32). Afin d'éviter toute perturbation, le Gate du mosphet est tiré vers la masse par le biais d'un résistance pulldown.

Pour activer le buzzer, l'ESP32 exite le Gate du mosphet pour le saturer de façon fréquentielle.

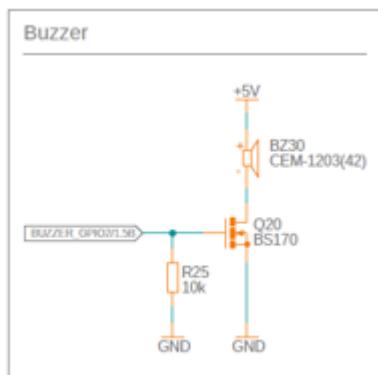


Figure 50 : SCHEMA Buzzer - PCB Mono-Protocole Zigbee

J'avais parlé plus haut de jumpers, les 4 PCBs les incluent tous au niveau du bus I²C & CAN (voir annexes 2, 9, 16 et 23).

Un jumper CAN branché dans un PCB signifie que ce dernier est le dernier périphérique ou le premier de tout le CAN BUS.

Si, en revanche, un des PCB du CAN BUS n'est ni dernier ni premier, alors l'utilisateur devra désacoupler le jumper, sinon tous les PCB qui le suivent seront ignorés par les signaux CAN.

Si l'utilisateur souhaite placer un élément externe depuis l'extension I²C, il le branchera depuis le connecteur suivant (voir annexes 2, 9, 16 et 23) :

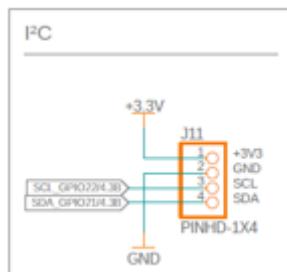


Figure 52 : SCHEMA I²C connecteur - PCB Mono-Protocole Zigbee

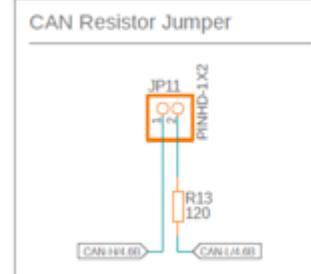


Figure 51 : SCHEMA Bus Jumpers - PCB Mono-Protocole Zigbee

Si cet élément inclut déjà des résistances pullup, alors l'utilisateur devra retirer les jumper des résistances I²C du PCB INDUSTRY 4.0 pour éviter qu'il y ait deux paire de pullup dans le bus.

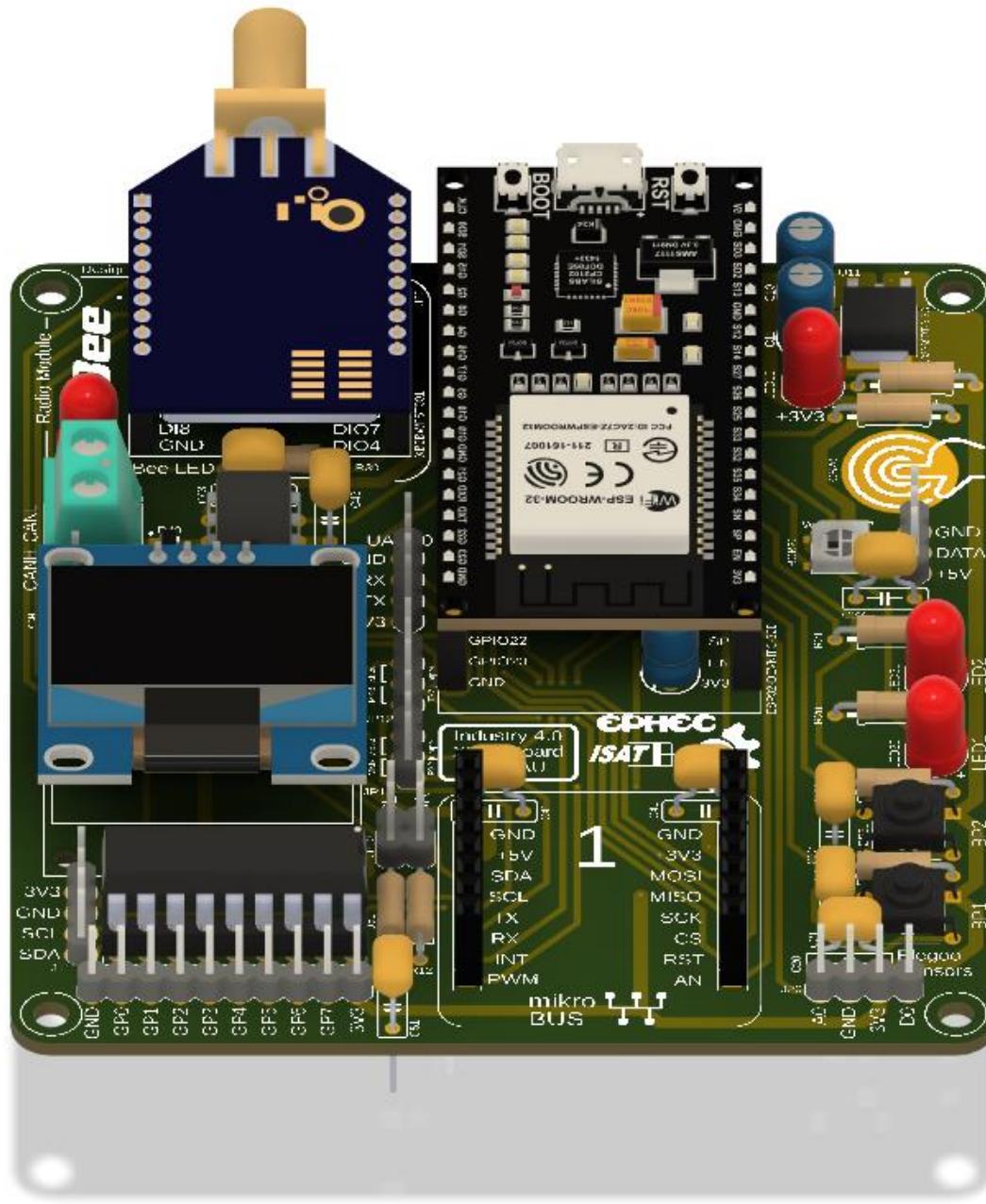
3.2.2.4. Rendu final

Figure 53 : 3D - PCB Mono-Protocole Zigbee

On remarque très vite que ce PCB Mono-Protocole orienté Zigbee est plus petit que celui Multi-Protocole. Cette impression est dûe au fait que l'emplacement mikrobus prive une bonne partie des GPIO de l'ESP32 d'ajouter d'autres capteurs ou actionneurs.

Il serait possible d'ajouter d'autres de ces éléments en rajoutant des jumper. Mais l'idée de conception de ce PCB est principalement celle d'être un nœud Zigbee sans trop aller plus loin dans l'ajout d'autres périphériques électroniques.

Les deux autres PCBs (orienté LoRa & SigFox) sont basés sur le même principe. Vous remarquez très vite qu'ils se ressemblent car je me suis appuyé sur la même architecture.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.2.3. PCB Mono-Protocole orienté LoRa**

Le PCB mono-protocole orienté LoRa est autant polyvalent que le précédent (PCB Zigbee). Il mesure 85mm x 95mm également. Il n'embarque pas exactement les mêmes périphériques, mais constitue une bonne partie de capteurs identiques.

Les modules de communications standardisés que le PCB embarque sont ceux des protocoles CAN, I²C, Wi-Fi, Bluetooth, Infrarouge et enfin, le LoRa évidemment.

Le bus I²C gère toujours l'OLED, & le MCP23008. Le protocole UART dirige le module CAN uniquement. Quant au SPI, c'est lui qui vient initialiser le module LoRa.

Au niveau des périphériques de type analogique, en plus du capteur d'Elegoo, on retrouve une LDR.

Du côté des périphériques digitales, le capteur d'Elegoo est également présent, les LEDs et les boutons poussoirs pareillement.

Et enfin, en simulant des protocoles depuis les broches digitales, on retrouve le protocole One Wire, permettant au capteur DS18B20 de récupérer la température à partir d'une seule GPIO. On retrouve également les émetteurs et récepteurs infrarouges connectés.

Bref, voici un résumé des périphériques qu'inclut le PCB mono-protocole orienté LoRa sous forme de tableau :

Analogique	Digitale	SPI	UART	I²C	Autre
Mikrobus	Mikrobus	Mikrobus	Mikrobus	Mikrobus	Mikrobus
LDR	LED1&2 BP1&2	LoRa	CAN	OLED MCP23008	Infrarouge DS18B20

3.2.3.1. Schema bloc

Figure 54 : Schema bloc - PCB Mono-Protocole LoRa

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.03.2.3.2. ESP32 Pinout

Protocol LoRa V1.1		
GPIO	Devices	Notes
0	LORA-NSS	SPI Chip Select
1	/	
2	LORA-DIO0	
3	/	
4	CAN-RX	
5	CAN-TX	
12	IR-Sender	
13	MKB-RST	MikroBus
14	LORA-RST	
15	MKB-INT	MikroBus
16	MKB-RX	MikroBus+UART2
17	MKB-TX	MikroBus+UART2
18	LORA+MKB-SCK	MikroBus+SPI Clock
19	LORA+MKB-MISO	MikroBus+SPI MISO
21	SDA (I_C)	Devices: OLED; MCP23008; MKB-SDA,SCL(MikroBus)
22	SCL (I_C)	
23	LORA+MKB-MOSI	MikroBus+SPI MOSI
25	DS18B20	
26	MKB-PWM	MikroBus
27	IR-Receiver	
32	LED2	LED ON when HIGH
33	LED1+MKB-CS	LED ON when LOW; Mikrobus+SPI Chip Select
34	BP1	HIGH when pushed
35	BP2	HIGH when pushed
36	MKB-AN	MikroBus
39	LDR	

Très similaire au PCB précédent, celui-ci a l'avantage de n'embarquer aucun jumper, offrant ainsi une utilisation simplifiée.

En contrepartie, il ne peut pas accueillir de capteurs de la série d'eleGO. La LDR remplace son emplacement. En effet, le câblage du module LoRa demande 3 GPIO digitale en plus du SPI, ce qui limite l'utilisation de certains éventuels capteurs, comme celui d'eleGO qui lui demandait deux GPIO.

L'ajout du DS18B20 vient rattraper cet inconvénient aussi. En effet, il n'est branché seulement depuis un GPIO sous le protocole One Wire.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.2.3.3. Schémas électroniques**

Le principe de pont diviseur de tension est présent dans la quasi-totalité des PCBs. Etant simple à la conception, et fiable dans l'utilisation, ce pont diviseur permet de générer en sortie une tension proportionnelle aux valeurs des résistances sur base d'une différence de potentiels initiale.

Je reprends l'exemple d'un cas utilisé dans ce PCB orienté LoRa : la LDR (représenté ci-contre) (voir annexe 17). La différence de potentiels initiale entre le GND et le 3.3V correspond à 3.3V.

Entre ces deux potentiels, je viens brancher deux résistances en série, dont une qui est la LDR.

Admettons maintenant que ces deux résistances aient la même valeur. En suivant l'équation proportionnelle du diviseur de tension, en sortie, nous obtenons une valeur de $U_{\text{sortie}} = 1.65V$:

$$U_{\text{sortie}} = U_{\text{initiale}} * \frac{LDR}{R24 + LDR}$$

A noter que le signal de sortie du pont ne dépassera pas 3.3V.

En simulant donc le fait que la valeur de la LDR varie en fonction de la luminosité ambiante, plus il fait sombre, donc plus sa résistance augmente, alors la tension lue en sortie augmente.

Etant donné que la tension varie, on peut la considérer comme étant de type analogique. L'ESP32 fait donc appel à l'interface ADC pour lire ce signal et le convertir en numérique afin de l'interpréter numériquement sous 12 bits.

Le DS18B20 présenté ci-contre (voir annexe 17) détient une particularité unique en son genre grâce à son protocole One Wire.

Ainsi, grâce à ce protocole, il est possible de gérer une bonne quantité de capteurs du même genre depuis un GPIO commun.

Le signal a besoin d'une résistance pullup pour éviter tout interférence externe.

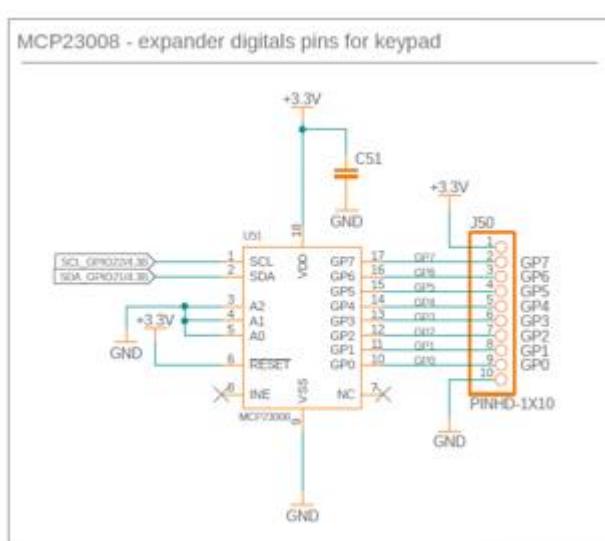


Figure 55 : SCHEMA LED - PCB Mono-Protocole LoRa

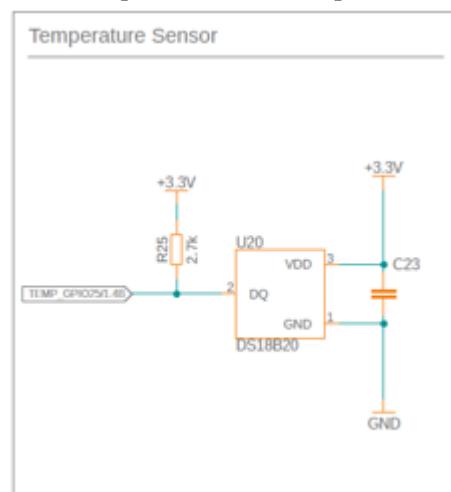


Figure 56 : SCHEMA DS18B20 - PCB Mono-Protocole LoRa

Le MCP3008 est un périphérique I²C. L'adresse I²C que je lui ai fourni sous 3 bits est 000. En effet, les pins A0 à A2 sont tirés vers la masse (voir annexe 20).

Il est possible de détecter tout évènement depuis ses sorties GP0 à 7 depuis la broche INE du circuit intégré. Cette dernière change d'état à chaque fois qu'une action a lieu au niveau de ses GP. En la liant donc à une GPIO de l'ESP32 en tant qu'interruption externe, l'ESP32 ne sera plus obligé de questionner en I²C si un évènement a lieu.

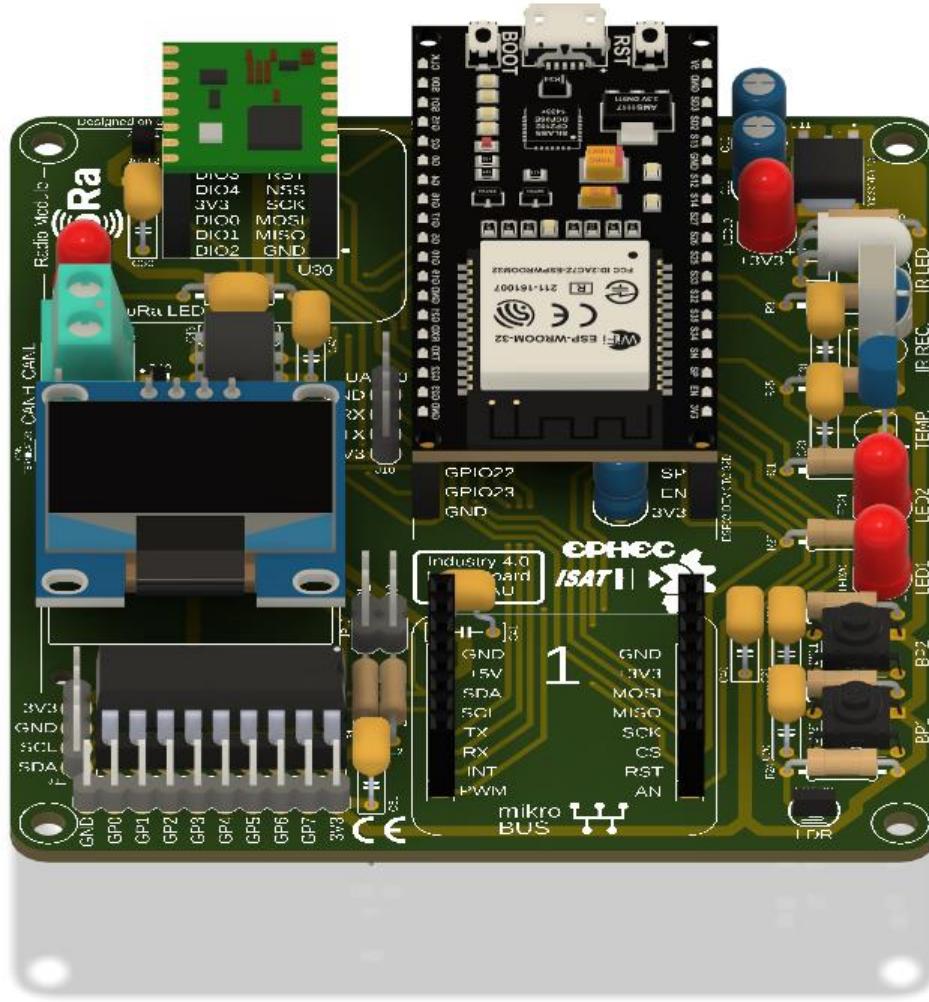
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.2.3.4. Rendu final**

Figure 57 : 3D - PCB Mono-Protocole LoRa

Le principe du routage a été un vrai défi dans tous les PCB. Un outil d’auto-routage est mis à disposition aux développeurs, mais je trouve personnellement que cela gâche le plaisir et la satisfaction de créer soi-même un PCB.

Le routage a donc été fait manuellement dans tous les PCBs (voir annexes 7, 14, 21 et 28). La section, ou plutôt la largeur des pistes cuivrées, est proportionnelle aux courants qui circuleront dans ces dernières.

J’ai utilisé des largeurs de 24mils (millième de pouce) pour les pistes de données, comme ceux des bus I²C, ou des boutons, etc... Du côté des pistes d’alimentations 3.3V / 5V, j’ai utilisé une largeur de 50mils.

Etant donné que dans un circuit électronique le signal GND est le plus fréquent, j’ai fait un plan de masse de la couche bottom du PCB. Cela augmente de l’espace dans la couche TOP pour y placer d’autres types de signaux.

Le courant tiré depuis la sortie 3.3V du régulateur est plus grand que le PCB. J’ai donc placé un plan de 3.3V pour favoriser la circulation locale du courant tout en faisant office de dissipateur thermique. En effet, le régulateur ayant le format SMD, est soudé sur cette surface.

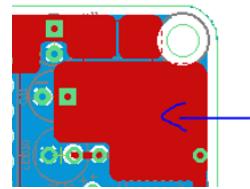


Figure 58 : ROUTAGE Plan de cuivre - PCB Mono-Protocole LoRa

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.2.4. PCB Mono-Protocole orienté SigFox**

Le PCB mono-protocole orienté SigFox est autant polyvalent que les deux précédents (PCB Zigbee & LoRa). Il mesure 85mm x 95mm également. Il n'embarque pas exactement les mêmes périphériques, mais constitue une bonne partie de capteurs identiques.

Les modules de communications standardisés qu'il embarque sont ceux des protocoles CAN, I²C, Wi-Fi, Bluetooth, et enfin, le SigFox évidemment.

Le bus I²C gère aussi l'OLED, & le MCP23008. Le protocole UART dirige le module CAN et le module Sigfox. Tout comme le module Zigbee, celui-ci communique par commande AT.

Au niveau des périphériques de type analogique, il n'y a que le capteur de la série d'Elegoo.

Du côté des périphériques digitales, le capteur d'Elegoo est également présent, les LEDs et les boutons poussoirs pareillement, mais aussi le capteur ultrasonique. Ce dernier occupe 2 GPIO de l'ESP32.

Et enfin, en PWM, un servo moteur est également prévu. Comme application, ce dernier peut, par exemple, être utilisé avec le capteur ultrasonique. Il se voit placé sur l'axe rotatif du moteur pour augmenter sa portée de détection angulaire.

Bref, voici un résumé des périphériques qu'inclut le PCB mono-protocole orienté LoRa sous forme de tableau :

Analogique	Digitale	SPI	UART	I ² C	Autre
Mikrobus	Mikrobus	Mikrobus	Mikrobus	Mikrobus	Mikrobus
Capteur Elegoo A0 & D0	BP1&2		CAN	OLED	Servo Motor
	LED1&2		SigFox	MCP23008	
	Ultrasonic S.				

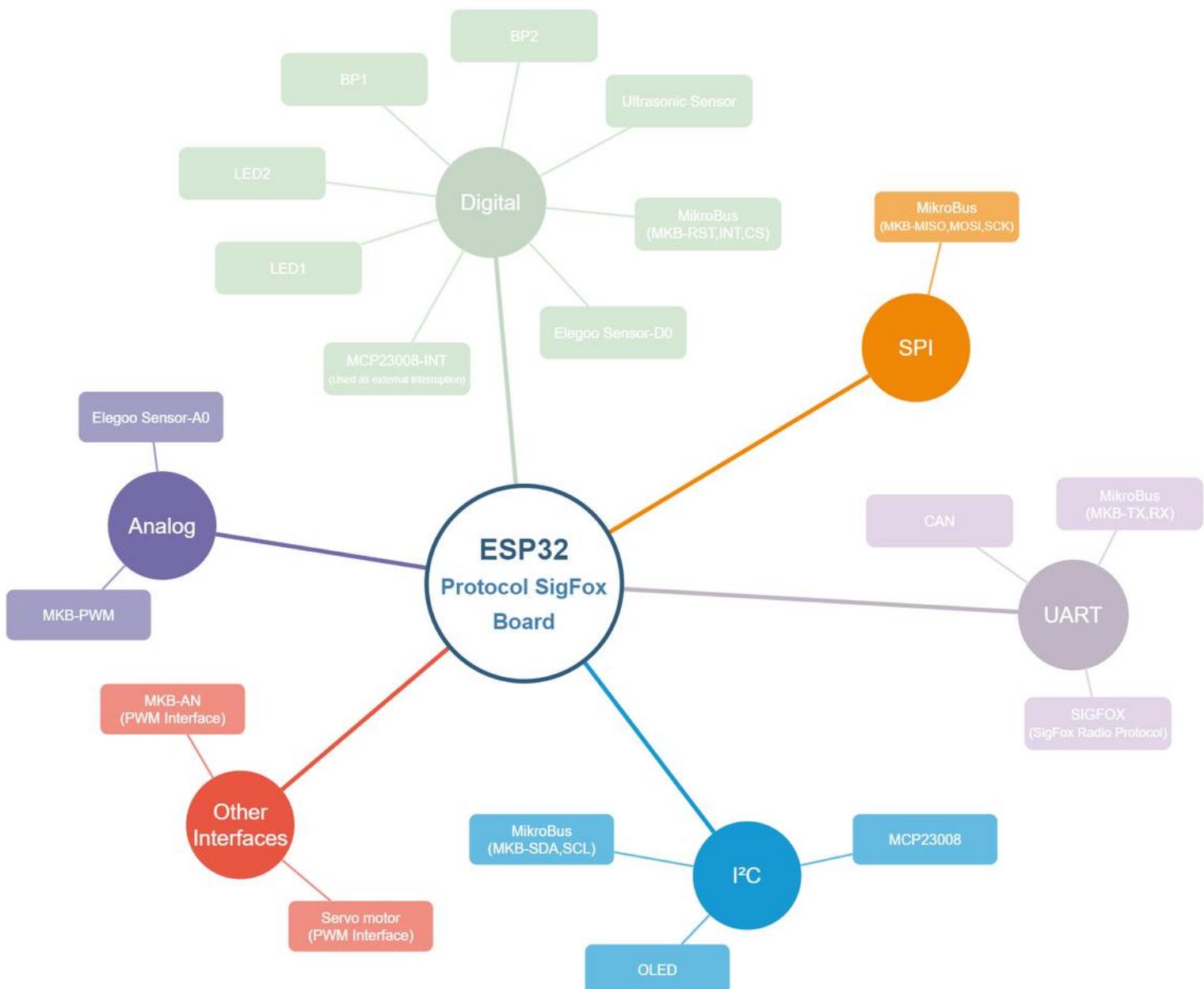
3.2.4.1. Schema bloc

Figure 59 : Schema Bloc - PCB Mono-Protocole SigFox

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

3.2.4.2. ESP32 Pinout

GPIO	Devices	Notes
0	MKB-RST	MikroBus
1	/	
2	MKB-INT	MikroBus
3	/	
4	CAN-RX	
5	CAN-TX	
12	TRIG (Ultrasonic)	
13	Servo	
14	ECHO (Ultrasonic)	
15	MPC23008-INT	
16	IO16 Jumper	Jumper JP12 : Choice of MKB-TX or Sigfox-RX; UART2
17	IO17 Jumper	Jumper JP13 : Choice of MKB-RX or Sigfox-TX; UART2
18	MKB-SCK	MikroBus+SPI Clock
19	MKB-MISO	MikroBus+SPI MISO
21	SDA (I^C)	Devices: OLED; MCP23008; MKB-
22	SCL (I^C)	SDA,SCL(MikroBus)
23	MKB-MOSI	MikroBus+SPI MOSI
25	E.Sensors-D0	
26	MKB-PWM	
27	LED1	LED ON when HIGH
32	LED2	LED ON when HIGH
33	MKB-CS	MikroBus+SPI Chip Select
34	BP1	HIGH when pushed
35	BP2	HIGH when pushed
36	MKB-AN	MikroBus
39	E.Sensors-A0	

Très similaire au PCB orienté Zigbee, ce PCB SigFox a également 2 jumper (voir annexe 23). Leur disposition est également dûe au fait de mapper l'UART2 vers le MikroBus, ou vers le module SigFox lui-même.

Dans certains de ces PCBs, il est important de noter que les LED1&2 ne sont pas constamment actives à l'état haut. En effet, pour économiser une GPIO, j'ai parfois branché une LED et un Chip Select sur une même GPIO. Etant donné que les Chip Select des modules SPI sont généralement inversées, alors la LED que j'ai placée est quant à elle également inversée, donc active à l'état bas. En somme, lorsqu'un périphérique SPI est appelé par L'ESP32, la LED s'allume.

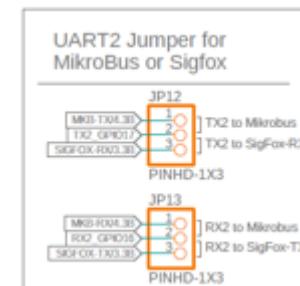


Figure 60 : SCHEMA Jumpers - PCB Mono-Protocole SigFox

3.2.4.3. Schémas électroniques

Certaines cartes ESP32 ne passent pas automatiquement en mode flash à l'instant où nous essayons de télécharger automatiquement :

DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM7

Figure 61 : Erreur de démarrage ESP32 - PCB Mono-Protocole SigFox

Une des solutions est de maintenir le bouton boot qui est placé sur l'ESP32 pour le forcer. Mais lorsque nous sommes en plein débogage de code, cela devient fastidieux à force d'appuyer à chaque fois. Il m'est cependant déjà arrivé que l'ESP32 ne se mette pas en mode flash même si j'appuyais sur le bouton adéquat.

J'ai alors cherché une solution à cet inconvénient, et il se trouve qu'en cablant un condensateur électrolytique de $10\mu F$ aux broches GND & EN de l'ESP32, ce dernier boot automatiquement (voir annexe 23).

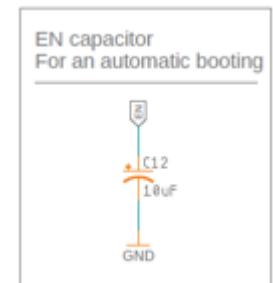


Figure 62 : SCHEMA condensateur EN - PCB Mono-Protocole SigFox

Je n'ai pas encore parlé du branchement des LEDs jusqu'à maintenant (voir annexe 24). Celles du PCB SigFox sont toutes actives à l'état haut du GPIO concerné. En respectant donc bien leur polarité, la cathode doit être en direction du GND et à l'anode c'est du coté de la source de tension, ou plutôt la GPIO de l'ESP32.

Le choix de la résistance à placer en série dépend du courant que doit consommer la LED, de la tension de seuil de la LED, et de la source de tension initiale.

Une LED verte a une tension de seuil légèrement supérieure à 1.8V. L'intensité de la lumière que je souhaite avoir, directement proportionnelle à l'intensité de la LED qui la traverse, correspond à approximativement 5 à 10mA. Et du coté de la source de tension initiale, nous avons à disposition une tension logique de 3.3V.

$$R = \frac{3.3V - 1.8}{0.0075A} = 200 \Omega$$

Ainsi, en faisant la loi d'Ohm pour trouver la valeur de la résistance à placer en série à la LED nous obtenons une valeur de 200Ω . J'avais à disposition une résistance assez proche, de 330Ω . Cela a fait l'affaire.

Du coté du branchement des boutons, on s'aperçoit aussi qu'il y a une résistance. Mais elle n'a pas du tout cette fonctionnalité de limitation de courant.

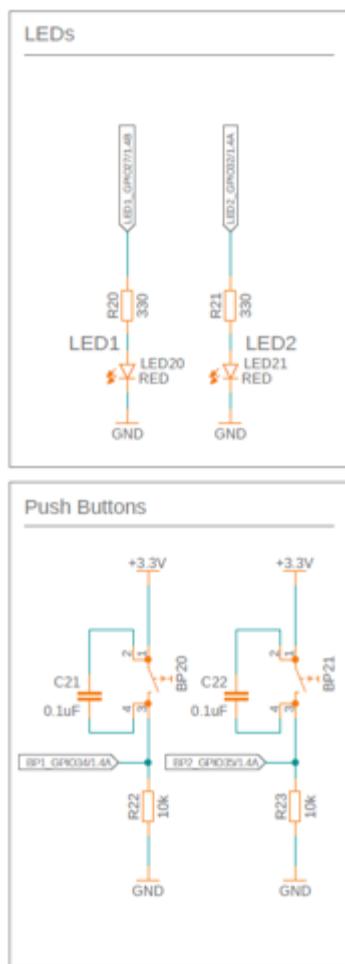


Figure 63 : SCHEMA LEDs et Boutons - PCB Mono-Protocole SigFox

Si on branchait bêtement le bouton sur une GPIO, il y aurait des interférences lorsque le bouton est relâché. En effet, étant donné que cette GPIO est en mode lecture uniquement, la tension qui se trouve sur la GPIO est volatile. Il y aurait donc détection de tout électricité statique. Cela pourrait se comporter comme une antenne.

Pour résoudre ce problème, on force l'état au niveau bas quand le bouton est relâché par le biais d'une résistance pulldown, et lorsqu'il est appuyé, le 3.3V se redirige vers le GPIO.

Cela dit, on se rend compte très vite, qu'à cause de la résistance, le circuit consomme du courant quelque soit l'état du bouton. La valeur de la résistance choisie pour ce type d'application s'élève à $4.7\text{K }\Omega$ ou $10\text{k }\Omega$ en général. Ces valeurs ont été choisies dans le but d'avoir un courant ni trop minime, ni trop élevé du côté de la GPIO du microcontrôleur, l'ESP32. En effet, cette plage de courant correspond à une détection d'état haut ou bas de façon fiable.

3.2.4.4. Rendu final

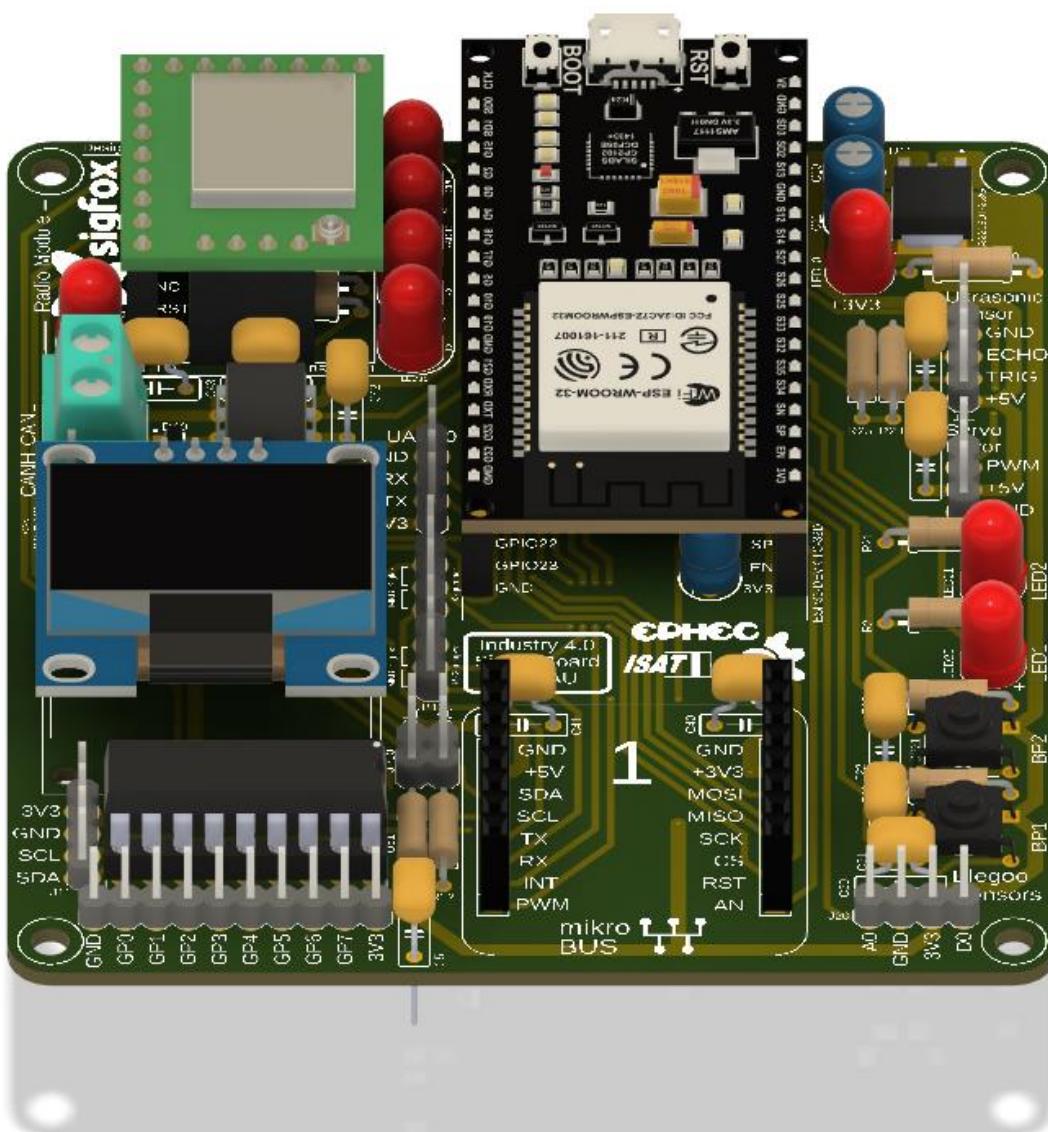


Figure 64 : 3D - PCB Mono-Protocole SigFox

3.2.5. Anciennes versions des PCBs

Vous avez probablement remarqué, les PCBs que j'ai présentés ci-dessus sont actuellement les dernières versions. Ce qui signifie que des anciennes versions ont été créées.

3.2.5.1. PCB Multi-Protocole (V1.0 à V2.0)

L'ancienne version du PCB Multi-Protocole était entièrement fonctionnelle. Je suis passé d'une V1.0 à une V2.0. Ce qui signifie que de la première à la deuxième version, des modifications de type « amélioration » ont été réalisées. S'il s'agissait d'une modification de type « minime » ou « débogage », j'aurais fixé la M&J à V1.1, mais ce n'est pas le cas avec ce PCB.

Son ancienne version mesurait 100m x 100m. Elle n'équipait pas d'emplacement RFID. Etant donné que le seul bus SPI était localisé du côté du module LoRa, j'ai placé les headers femelles du RFID entre l'ESP32 et le module LoRa. Car le RFID est commandé en SPI. J'ai donc dû élargir le PCB de 6mm pour qu'il y ait assez d'espace pour les pistes, et pour les interférences mécaniques entre le RFID et l'ESP32.

Avant, sous le capteur de présence PIR, il n'y avait que deux résistances. Elles permettaient de diviser la tension sortie de son signal de présence en 3.3V, étant donné que ce capteur est alimenté en 5V. Mais par pur hasard pendant mes recherches documentaires, je me suis rendu compte que la sortie du signal de détection était de 3.3V. J'ai alors supprimé ce pont diviseur de tension inutile pour libérer de l'espace en dessous.

L'ancienne version de ce PCB Multi-Protocole n'avait pas de module CAN. Et donc, suite à cette modification, je me suis précipité à ajouter le module CAN en dessous du module PIR. En effet, après avoir enlevé les résistances, de l'espace s'est libéré en dessous du module PIR.

Du coté des sorties MCP23008, il n'y avait que ces GPx. J'ai ajouté des broches d'alimentations GND & 3.3V sur la même lignée des connecteurs des GPx.

Dans la V1.0, je me suis rendu compte que j'avais oublié de placer une extension du bus I²C sous forme de headers femelles. Je l'ai rajoutée à la V2.0.

Les autres modifications sont minimes. Certaines sont quelques arrangements de pistes, d'autres de déplacement & ajout d'images & logos.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

3.2.5.2. *PCB Mono-Protocole orienté Zigbee (V1.0 à V1.1)*

Le PCB orienté Zigbee a eu un problème. Lorsque j'essayais de télécharger un code, le buzzer se mettait à grésiller sans que l'ESP32 se mette en mode flash ou de téléchargement. Il fallait que je retire l'ESP32 de son emplacement pour pouvoir uploader en paix.

Tout au long du développement de mon TFE, je me suis longtemps basé sur un tableau reprenant certaines caractéristiques spécifiques des GPIO de l'ESP32. Comme précisé précédemment, on ne peut pas brancher un périphérique n'importe où dans l'ESP32. Dans l'annexe 1 se trouve justement ce tableau : **Restrictions des pinout de l'ESP32**.

Cependant, suite aux recherches effectuées, je me suis rendu compte qu'une donnée cruciale manque dans ce tableau, celle-ci :

Table 3: Strapping Pins

Voltage of Internal LDO (VDD_SDIO)					
Pin	Default	3.3 V	1.8 V		
MTDI	Pull-down	0	1		
Booting Mode					
Pin	Default	SPI Boot	Download Boot		
GPIO0	Pull-up	1	0		
GPIO2	Pull-down	Don't-care	0		
Enabling/Disabling Debugging Log Print over U0TXD During Booting					
Pin	Default	U0TXD Active	U0TXD Silent		
MTDO	Pull-up	1	0		
Timing of SDIO Slave					
Pin	Default	FE Sampling FE Output	FE Sampling RE Output	RE Sampling FE Output	RE Sampling RE Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

Figure 65 : Strapping Pins

Ce tableau représenté ci-dessus nous précise justement les restrictions à respecter lors de booting du microcontrôleur. Il explique que ce mode de téléchargement fonctionne uniquement si le GPIO0 & le GPIO2 sont tirés vers le bas, vers le GND. Or, le GPIO2 représenté dans le tableau dont je me suis basé tout le long du projet (voir annexes 1) n'indique aucune condition ou éventuelles précisions.

Le périphérique MCP23008 avait sa broche d'interruption qui était connecté au GPIO2 justement dans l'ancienne version. Et il se trouve que dans ce circuit intégré, une résistance de type pull-up y est placée. Cette dernière vient déranger le booting parce que les conditions d'avoir le GPIO0 & GPIO2 à l'état bas ne sont pas respectées.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Avant, le périphérique MCP23008 avait sa pin d'interruption à la broche GPIO2 de l'ESP32. Quant au périphérique Buzzer, il était connecté à la GPIO0. Ces deux périphériques sont inversés actuellement. J'ai donc corrigé cela plus proprement à la V1.1 de ce PCB.

Mais en attendant la livraison de sa V1.1, j'ai switché les GPIO de ces deux périphériques par le biais de pontages en dessous du PCB :

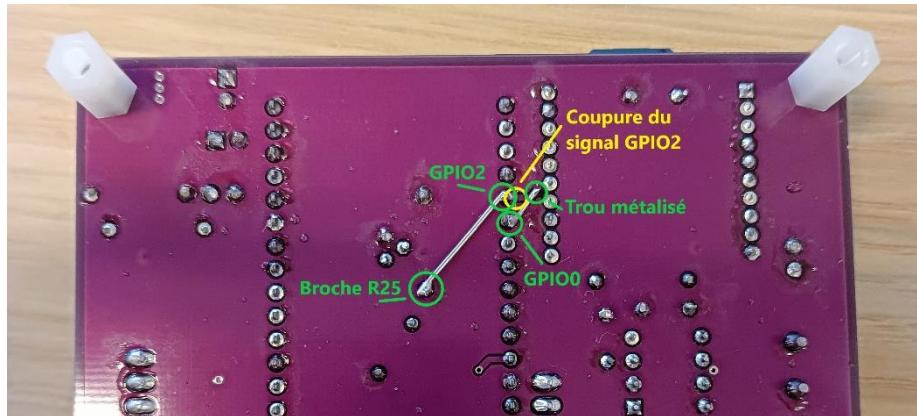


Figure 66 : Pontage du PCB Mono-Protocole Zigbee

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

3.2.5.3. PCB Mono-Protocole orienté LoRa (V1.0 à V1.1)

Ce PCB Mono-Protocole orienté LoRa n'avait qu'un seul problème. Il était facile de le corriger.

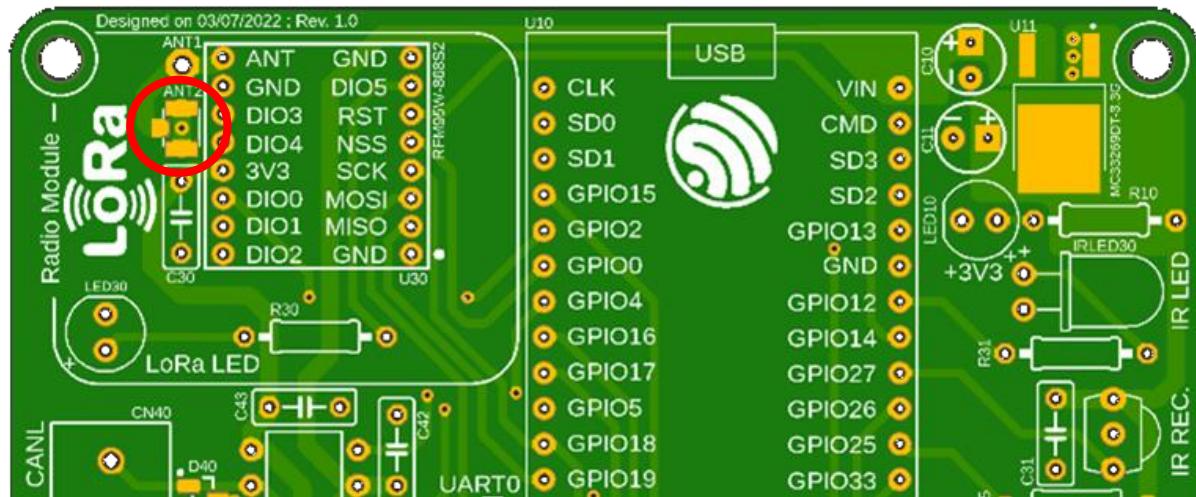


Figure 67 : Trou métallisé mal placé dans le PCB Mono-Protocole LoRa

Ce qui est entouré en rouge correspond au connecteur SMD SMA. Il s'agit d'un connecteur permettant d'insérer une antenne. Au centre du footprint (l'empreinte) du connecteur se trouve un trou métallisé.

Ce trou métallisé peut court-circuiter le GND avec le signal de l'antenne si le connecteur SMA vient à se souder dessus. Cela provoquerait en conséquence une antenne tirée vers la masse. Elle ne détectera aucune fréquence.

La solution est de surélever le connecteur pendant qu'on le soude afin qu'il n'y ait pas contact entre le trou métallisé et l'armature inférieure du connecteur.

Dans la V1.1 j'ai supprimé ce trou métallisé.

Il se trouve que ce PCB a également eu un autre élément qui devait être changé. Il s'agissait de l'emplacement des trous de fixations de l'OLED. Ils étaient décalés et ne correspondaient pas mécaniquement à ceux de l'OLED lui-même. J'avais cependant déjà corrigé cela avant de passer à l'achat des PCBs.

En fait, par erreur, lorsque l'on a acheté ce PCB, nous nous sommes trompés de version. Cette dernière avait la version de l'OLED ancienne...

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.2.5.4. PCB Mono-Protocole orienté SigFox (V1.0 à V1.1)**

Du côté du PCB Mono-Protocole orienté SigFox, le même problème que celui Zigbee était présent. L'ESP32 ne se bootait pas lorsque nous essayons de télécharger un code.

J'ai changé cela par le biais de pontages (en vert) en coupant le signal GPIO15 (en jaune). J'ai switché la broche GPIO2 (interruption du MCP23008) avec la broche GPIO15, qui correspond à la pin d'interruption du mikrobus :

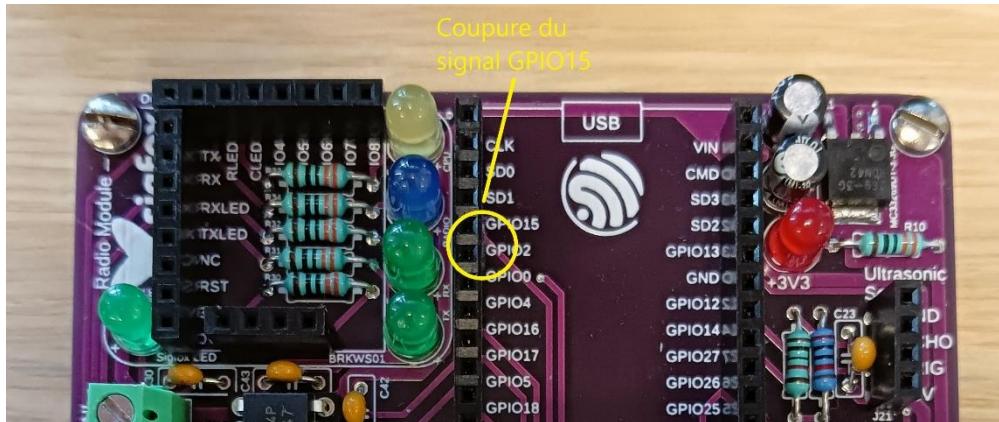


Figure 68 : Coupure d'un signal dans le PCB Mono-Protocole LoRa

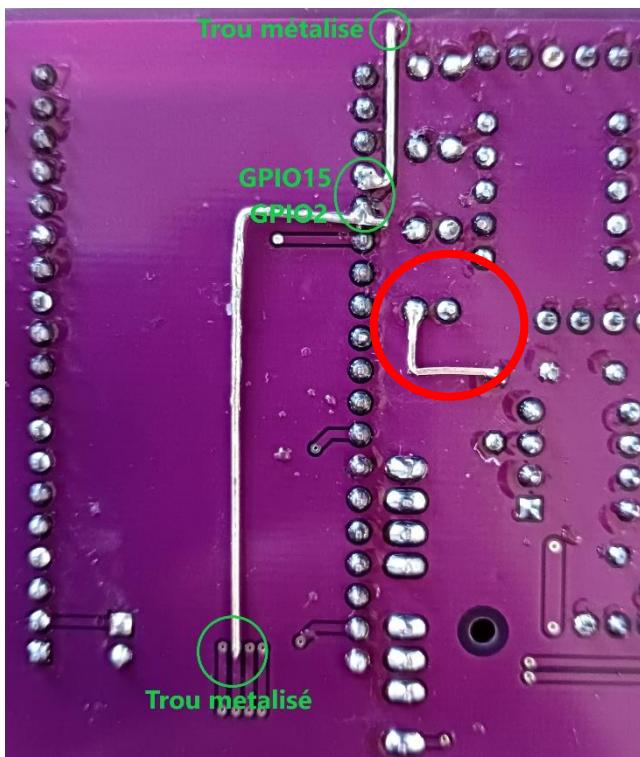


Figure 69 : Pontage du PCB Mono-Protocole LoRa

Vous pouvez cependant remarquer que ce qui est entouré en rouge c'est un autre pontage.

Ce dernier correspond aux changements de sens de toutes les LEDs (signal, CPU, TX, RX) du module SigFox.

Initialement, étant donné que le datasheet du module SigFox est incomplet, le maître de stage a supposé que ces sorties de LEDs s'activaient à l'état bas.

Mais une fois que les PCBs ont été livrés, après les tests, On voit que les LEDs s'activent au niveau haut.

J'ai alors inversé les LEDs, puis changé l'arrivée du 3.3V par le GND.

Ce problème a été présent aussi dans le PCB Multi-Protocole dans sa version 1.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

3.2.6. Supports imprimés en 3D

Les supports imprimés en 3D permettent de fixer certains composants qui ont tendance à se déconnecter au fur et à mesure que nous manipulons énergiquement les différents PCBs.

Etant donné que les étudiants manipuleront ces PCBs les années qui suivent, une rigidification de ces derniers doit être effectuée. En moyenne, il y a 10 groupes de binôme par année, soit 10 copies de chaque PCB pour une classe tout entière.

L'accéléromètre a tendance à se balancer lorsqu'on le branche sur les headers femelles. Voici ci-contre son support. J'en ai imprimé 10. En effet, ces accéléromètres ne sont localisés qu'une fois dans le PCB Multi-Fonction parmi les quatre autres PCB. Ses trous de fixation sont prévus pour y insérer des vis M3 suivis d'écrous.



Figure 70 : Support MPU6050



Le support présenté ci-contre correspond à celui de l'OLED. J'en ai imprimé 30. Il sera présent dans tous les PCBs excepté celui Multi-Protocole. Ses trous de fixation sont prévus pour des vis M2 avec des écrous.

Figure 71 : Support OLED (4 trous)

Le support présenté ci-contre à droite est également celui de l'OLED. J'en ai imprimé 10. Il est présent dans les PCBs Multi-Protocole.

Comparé au support précédent, celui-ci n'est équipé que de deux trous de fixation au lieu de 4. En effet, pour des raisons mécaniques, un 4^{ème} « bras » de fixation n'a pas pu être installé.



Figure 72 : Support OLED (2 trous)

La raison mécanique est que l'OLED est placé dans le coin inférieur droit du PCB (voir annexe 8). Son footprint rentre donc en collision avec les 4 trous de fixation principaux du PCB. Regardez l'image ci-contre.

J'ai été amené à créer un footprint sur mesure et personnalisé pour qu'il n'y ait pas également de superposition de lignes blanches entre le cercle du trou de fixation principale du PCB et le cadre de l'OLED.

Figure 73 : Encombrement de l'OLED dans le PCB Multi-Protocole

Quant au dernier support, il s'agit du capteur de présence PIR. Ce dernier a été le plus complexe. Puisqu'il y a le module CAN et quelques périphériques électroniques en dessous, il ne fallait surtout pas entrer en collision mécanique.

J'en ai imprimé 10. Il est présent dans les PCBs Multi-Protocole. Ce capteur est le plus « volatile » comparé aux autres modules ayant un support. En effet, il n'est branché que depuis 3 broches.

Ce support est équipé de 4 trous. Les deux inférieurs seront constamment fixés par des vis M2 et écrous. Les deux trous supérieurs sont dimensionnés pour qu'ils accueillent des vis M1.5 depuis le capteur lui-même. Ces dernières s'enlèvent avec le capteur dans le cas où nous essayons d'atteindre les jumper du module CAN.



Figure 74 : Support PIR

3.3. Mise en œuvre de la solution software

3.3.4. Explication de la programmation des périphériques & GitHub

Chaque périphérique a été soigneusement programmé. J'ai ouvert un dépôt de GitHub où mes codes sources sont placés (<https://github.com/DavideDiVenti/TFE-3AU-2022>). Je ne développe pas l'explication de chaque code sur ce rapport, mais tout est expliqué sur le GitHub.

Là se trouve une documentation permettant aux étudiants de dompter plus facilement les différents PCBs. Dans cette documentation se trouvent les codes sources des différents périphériques. Au points suivants (3.3.5 & 3.3.6) se trouvent le TP Industrie 4.0 et aussi tous leurs codes sources qui y sont déposés.

3.3.5. Explication de la solution du TP Industrie 4.0 : TP Zigbee2MQTT

Ce point 3.3.5 est une grande parenthèse du thème. Il s'agit de la première version du TP que j'ai réalisée. J'ai rencontré un souci vers la fin du développement du TP qui m'a empêché de le finaliser.

Introduction :

La fonctionnalité Zigbee2MQTT peut s'implémenter sous forme de container docker pour opérer à une application Zigbee vers MQTT. L'étude que j'ai effectuée dessus a été assez approfondie malgré le fait qu'elle n'a pas pu être entièrement finie. Cela dit, bien que je n'aie pas pu finaliser l'approche Zigbee2MQTT pour des raisons de ressources limitées sur le net, je tiens quand même à développer cette étude ultérieurement.

Le problème a été le suivant : les modules Zigbee placés dans les PCBs ne sont pas entièrement compatibles avec la bibliothèque de composants de la passerelle Zigbee2MQTT. Et de ce fait, l'acquisition intégrale des données utiles voyageant dans le réseau maillé local de nos PCBs, est grandement limitée. Si les modules Zigbee des PCBs avaient été différents et compatibles, la dernière procédure que j'ai effectuée aurait fonctionné avec succès : faire des acquisitions et actions sur les périphériques Zigbee du réseau maillé local.

Je ne regrette rien, j'ai malgré tout appris beaucoup de choses. Anticiper le fait que mes modules ne soient pas compatibles au moment où je les ai placés dans la liste de matériel à acheter m'était impossible au stade où j'étais.

Dongle Zigbee2MQTT :

Bref, reprenons depuis le début. Comme matériel à disposition pour commencer le développement de ce TP, j'avais mes PCBs, mon PC portable Windows 10 et le dongle. Ce dernier est un *Sonoff Zigbee 3.0 USB Dongle Plus* intégrant un firmware pré-flashé CC2652P comme présenté ci-contre :



Figure 75 : Dongle Zigbee2MQTT

Il s'agit d'un dongle entièrement adapté pour les applications de type Zigbee2MQTT. Il est préconçu pour agir en tant que coordinateur une fois infiltré dans le réseau domotique maillé local Zigbee. Une fois configuré, il serait capable d'observer et d'agir sur les différentes trames de charges utiles des différents noeuds Zigbee. Il serait également capable de définir qui est un *endpoint* et un *router* (prochainement expliqué), le tout depuis un *frontend*.

Configuration du conteneur :

Mais pour en arriver à ce stade, plusieurs étapes sont nécessaires. Les premières étapes, vous les connaissez. La première est celle d'installer docker (je l'avais fait sur mon PC), et la deuxième est celle d'y placer un container MQTT. Etant donné que j'étais sous Windows 10, il m'a fallu installer linux dans le PC. J'ai mis Debian. Pour installer docker, j'ai dû ouvrir le powershell du Debian et entrer la commande de téléchargement en WSL.

Une fois tout cela fait, un nouveau conteneur peut être placé, celui Zigbee2MQTT. Je me suis basé sur un tutoriel (<https://www.youtube.com/watch?v=6MiU98mQQTQ&t=1002s>) pour suivre les étapes de téléchargement et configuration.

Connecté en WSL, la première commande à introduire est celle de créer un répertoire où les fichiers de configuration ont eu lieu et d'y rentrer :

```
pi@raspberrypi : ~$ mkdir z2m
pi@raspberrypi : ~$ cd z2m/
```

Une fois entré dans le dossier, nous créons notre fichier docker-compose. La commande *sudo* est utilisée pour agir en tant qu'administrateur. La commande *nano* utilisée agira comme un éditeur de texte. Ce docker-compose est un outil pour définir des applications multi-conteneurs. Il sera lu et exécuté lorsque nous entrons une commande spécifique. Il permettra d'exécuter et faire tourner les différentes configurations des différents conteneurs du docker.

```
pi@raspberrypi : ~/z2m$ sudo nano docker-compose.yml
```

Nous serons ensuite dirigés vers une nouvelle interface, celle de l'éditeur *nano*. Elle sera vide au départ. Il nous suffira d'écrire ces paramètres :

```
GNU nano 5.4                                            docker-compose.yml

version: '3'
services:
  zigbee2mqtt:
    container_name: zigbee2mqtt
    image: koenkk/zigbee2mqtt
    volumes:
      - /opt/data/z2m:/app/data
      - /run/udev:/run/udev:ro
    devices:
      - /dev/ttyUSB0:/dev/ttyUSB0
    restart: always
    network_mode: host
    privileged: true
    environment:
      - TZ=Europe/Brussels
```

Figure 76 : docker-compose Zigbee2MQTT

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Ensuite, pour sauvegarder et quitter les changements effectués, j'utilise CTRL+S puis CTRL+X. Nous pouvons ensuite vérifier s'il a bien été créé grâce à la commande `ls`. Cette commande permet l'observation du contenu d'un répertoire. En accédant ensuite à nos fichiers d'options à la racine du linux, nous pouvons voir qu'un fichier `configuration.yaml` et `log` a été créé :

```
pi@raspberrypi : ~/z2m$ ls
docker-compose.yml
pi@raspberrypi : ~/z2m$ cd /opt/data/z2m/
pi@raspberrypi : /opt/data/z2m$ ls
configuration.yaml      log
```

Nous devons éditer le fichier `configuration.yaml`. Une page d'éditeur nana s'affichera :

```
pi@raspberrypi : /opt/data/z2m$ sudo nano configuration.yaml
```

Dedans s'y trouveront des configurations similaires à ce qui est présenté ci-dessous, à quelques exceptions. Elles seront lues au démarrage du conteneur.

```
GNU nano 5.4                                     configuration.yaml
homeassistant: false
permit_join: true
mqtt:
  base_topic: zigbee2mqtt
  server: 'mqtt://192.168.137.122:1883'
serial:
  port: /dev/ttyUSB0
frontend:
  port: 8080
```

Figure 77 : configuration Zigbee2MQTT

L'option `permit_join` permet de savoir si des modules Zigbee ont librement le droit d'être détecté ou pas depuis le dongle Zigbee2MQTT.

L'option `mqtt` permet de configurer le topic correspondant (dans notre cas zigbee2MQTT) à notre application. Il faut de plus choisir l'adresse IP de notre broker MQTT au bon port (dans mon cas au port 1883).

L'option `serial` permet d'initialiser la communication UART au bon port USB correspondant au dongle Zigbee2MQTT.

Et enfin, l'option `frontend` permet à son tour d'ouvrir le port 8080 localement pour y insérer une interface utilisateur (prochainement expliqué).

Bref, une fois que tout est bien noté, on sauvegarde puis on quitte l'éditeur nano. On démarre ensuite le conteneur Zigbee2MQTT depuis le docker. En théorie, à ce stade, nous devons être en mesure de détecter n'importe quelle périphérique Zigbee depuis le port 8080, le frontend.

Débogage :

Mais, il y a toujours un mais, la théorie ne s'est pas appliquée de mon côté. Voici ce qu'il s'est passé :

```
Using '/app/data' as data directory
Zigbee2MQTT:info 2022-04-05 18:04:48: Logging to console and directory: '/app/data/log/2022-04-05.18-04-47' filename: log.txt
Zigbee2MQTT:info 2022-04-05 18:04:48: Starting Zigbee2MQTT version 1.25.0 (commit #6f1460e)
Zigbee2MQTT:info 2022-04-05 18:04:48: Starting zigbee-herdsman (0.14.20)
Zigbee2MQTT:error 2022-04-05 18:04:48: Error while starting zigbee-herdsman
Zigbee2MQTT:error 2022-04-05 18:04:48: Failed to start zigbee
Zigbee2MQTT:error 2022-04-05 18:04:48: Check https://www.zigbee2mqtt.io/guide/installation/20_zigbee2mqtt-fails-to-start.html for possible solutions
Zigbee2MQTT:error 2022-04-05 18:04:48: Exiting...
Zigbee2MQTT:error 2022-04-05 18:04:48: Error: Error while opening serialport 'Error: Error: No such file or directory, cannot open /dev/ttyACM0'
    at SerialPort.<anonymous> (/app/node_modules/zigbee-herdsman/src/adapters/z-stack/znp/znp.ts:146:28)
    at SerialPort._error (/app/node_modules/zigbee-herdsman/node_modules/@serialport/stream/lib/index.js:198:14)
    at /app/node_modules/zigbee-herdsman/node_modules/@serialport/stream/lib/index.js:242:12
```

Figure 78 : LOGS Dongle non reconnu

Il s'agit des logs du docker. On remarque très vite qu'une erreur a eu lieu, celle du fait que le dongle n'est pas reconnu par notre conteneur. Windows reconnaît le port USB du dongle mais du côté linux (donc le docker) ce n'est pas le cas.

L'explication de ce problème vous est sûrement évidente : sous un linux installé sur un pc Windows, la reconnaissance de périphérique externe est rejetée, je l'ignorais, même si cela paraît logique. Maintenant que je le sais. Cette machine virtuelle (le docker de mon pc) bloque donc l'accès au dongle. La preuve : l'erreur affichée dans les logs est identique quand j'enlève ou quand j'insère le dongle dans le pc malgré toutes les redirections de ports usb que j'ai effectuées dans les yml & yaml ("cannot open /dev/ttyACM0").

Recherches documentaires pour un éventuel débogage :

Je me suis alors tourné vers mon maître de stage. Il m'a pisté vers un lien du moins très intéressant. Ce lien explique les permissions d'accès d'un périphérique USB externe depuis un container linux. J'ai rencontré pas mal de soucis mais je suis parvenu à ajouter un USB device (disque clé USB 4Go) grâce à la procédure du lien. Le docker a réussi à setup l'USB sans problèmes. Par contre, si j'installe un appareil USB utilisant la communication serial (Arduino Uno, Dongle zeebee2MQTT...), le docker ne le reconnaît pas.

Beaucoup de forums précisent le fait que nous ne pouvons pas exposer un périphérique USB série ou similaire à la machine virtuelle exécutant des conteneurs Linux. Ce problème a persisté pendant plus de 4 ans apparemment. Un forum m'a été très instructif, et d'année en année on peut voir que les demandes de correction de ce problème s'agrandissaient : <https://github.com/docker/for-win/issues/1018>.

Très récemment, Microsoft a fait une M&J en ce qui concerne la gestion de périphérique entre Windows & linux. Depuis l'été dernier, on peut désormais connecter des périphériques USB à WSL 2. En d'autres termes, moi (et tous ceux qui étaient dans mon cas (quasi tout le monde)) qui n'avaient pas accès à l'emplacement équivalent du COM15 en WSL (comme ttyUSB15...), eh bien grâce à cette M&J je pourrai enfin le détecter. Ainsi, dans le fichier yml, j'insère le nouveau nom (ttyUSB15 par exemple) pour que le docker puisse setup la communication série du dongle. Voici là M&J : <https://devblogs.microsoft.com/commandline/connecting-usb-devices-to-wsl/>

Comme on peut le voir dans ce site, la M&J est adaptée à des PC Windows 11. Or, mon PC portable a un processeur intel de 3ème génération. Pour installer windows 11, il est requis que le PC soit équipé d'un processeur de 7^e génération minimum...

C'est là que l'idée d'acheter un Raspberry Pi 4 m'est parvenue. Elle réglerait ce problème car l'OS linux est nativement implémenté dans sa mémoire.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**L'après débogage :**

Le jour même où j'ai reçu le Raspberry à mon domicile (12 jours plus tard) je me suis directement précipité pour développer ce que j'avais préparé les jours précédents : le Zigbee2MQTT.

Grâce à la gestion physique USB entre le RPI et le dongle, le setup du dongle Z2M fonctionne enfin. Il est détecté au USB0 et il se connecte bien au MQTT du RPI4. En introduisant l'IP suivie du port 8080, j'ai le frontend qui s'affiche.

Dans ce frontend se trouve la liste de périphériques Zigbee connectés à proximité du dongle. Depuis cette liste, il est possible de voir toutes les configurations des modules zigbee (s'il est un routeur, un coordinateur, un endpoint, s'il est connecté ou déconnecté, etc..).

Sauf qu'à ce moment, ma liste était vide. Je n'avais connecté aucun module Zigbee encore, donc il n'y a pas de surprise pour le moment.

Avant de les mettre sous tension, les modules zigbee doivent être configurés sous la plateforme XCTU en tant que périphérique zigbee. XCTU est une application gratuite permettant aux développeurs d'interagir facilement avec n'importe quel module venant de la marque Digi RF.

Une fois que le module est connecté et est placé à proximité du dongle, on peut remarquer que dans le frontend s'affiche un nouveau périphérique dans la liste :

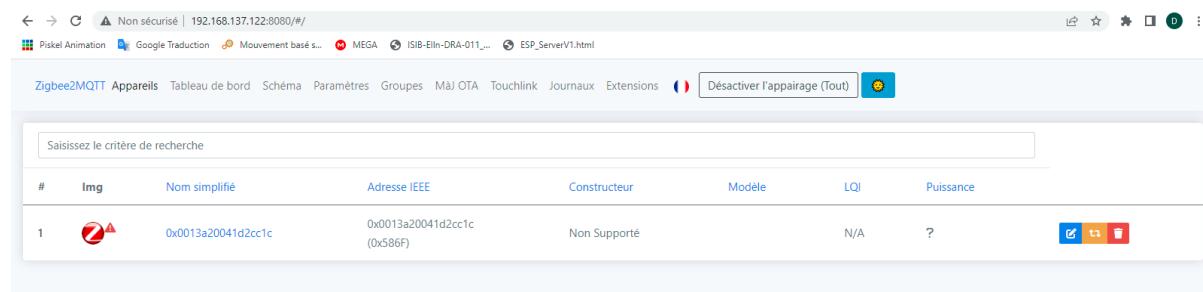


Figure 79 : XBee non reconnu

On remarque très vite qu'il y a un souci. Il n'est pas reconnu par le dongle. Et son initialisation a du mal à s'exécuter :

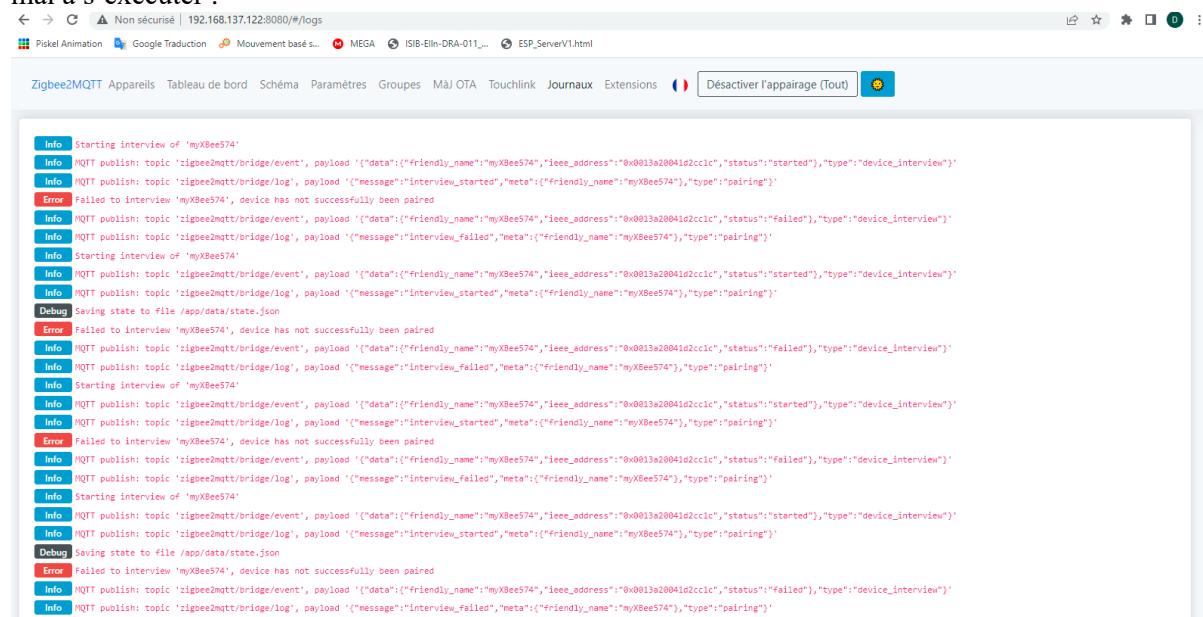


Figure 80 : LOGS XBee non reconnu par le dongle

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Je me suis donc mis à effectuer une recherche sur ce sujet. Apparemment, une clé de cryptage doit être insérée pendant la configuration de chaque module Zigbee sous XCTU. Le paramètre KY doit être fixé à la valeur 5A6967426565416C6C69616E63653039 :

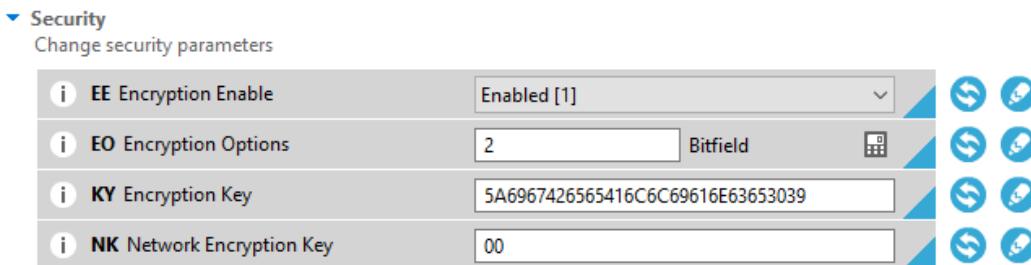


Figure 81 : Clé de chiffrement XCTU

A cet instant, le périphérique est automatiquement reconnu par le dongle :

```
Debug Device 'myXBee574' announced itself
Info MQTT publish: topic 'zigbee2mqtt/bridge/event', payload '{"data":{"friendly_name":"myXBee574","ieee_address":"0x0013a20041d2cc1c"},"type":"device_announce"}'
Info MQTT publish: topic 'zigbee2mqtt/bridge/log', payload '{"message":"announce","meta":{"friendly_name":"myXBee574"},"type":"device_announced"}'
Info Successfully interviewed 'myXBee574', device has successfully been paired
Info Device 'myXBee574' is supported, identified as: Digi Router (Xbee)
Info MQTT publish: topic 'zigbee2mqtt/bridge/event', payload '{"data":{"definition":{"description":"Router","exposes":[{"access":1,"description":"Link quality (signal strength)","name":"linkquality","property":"linkquality","type":"numeric","unit":"lqi","value_max":255,"value_min":0}], "model": "XBee", "options":[]}, "supports_ota":false, "vendor": "Digi", "friendly_name": "myXBee574", "ieee_address": "0x0013a20041d2cc1c", "status": "successful", "supported":true}, "type": "device_interview"}'
Info MQTT publish: topic 'zigbee2mqtt/bridge/log', payload '{"message":"interview_successful","meta":{"description":"Router","friendly_name":"myXBee574","model": "XBee", "supported":true, "vendor": "Digi"}, "type": "pairing"}'
Debug Saving state to file /app/data/state.json
Debug Saving state to file /app/data/state.json
```

Figure 82 : LOGS XBee reconnu par le dongle

J'ai alors configuré un deuxième module Zigbee, puis l'ai mis sous tension pour voir s'il était détecté également :

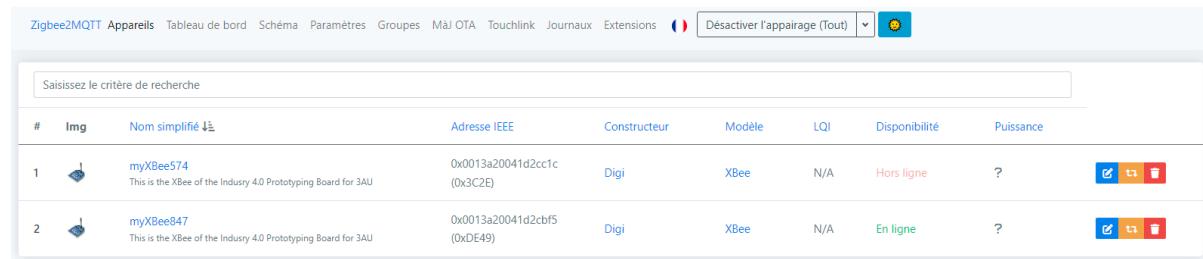


Figure 83 : Liste des XBee reconnus

Entre temps, le premier module s'est mis en veille, il s'est donc déconnecté. Mais à part ce détail, les deux modules sont détectés, et le premier maillage a pris forme :

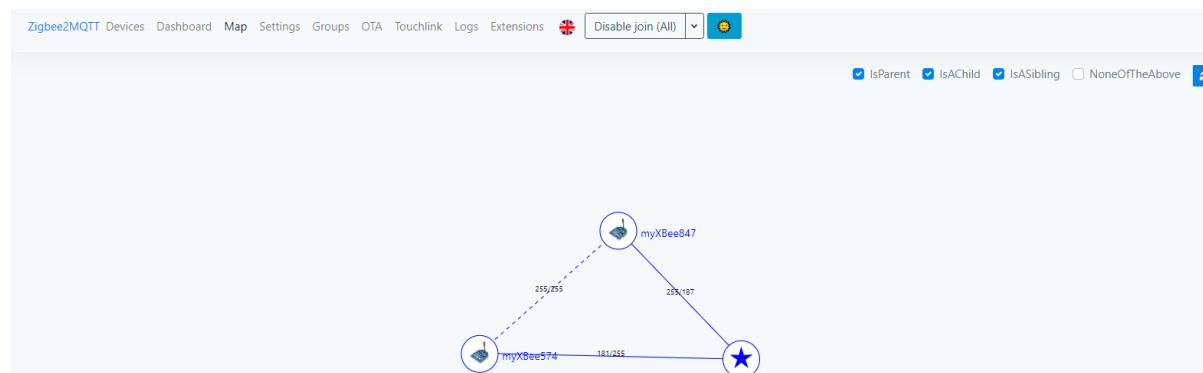


Figure 84 : Premier maillage Zigbee

L'étoile représenté correspond coordinateur, il s'agit du dongle Zigbee2MQTT.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Tout ceci est bien beau, mais malheureusement, en recherchant le XBee dans les périphériques supportés par Zigbee2MQTT (<https://www.zigbee2mqtt.io/devices/>), une mauvaise nouvelle est apparue :

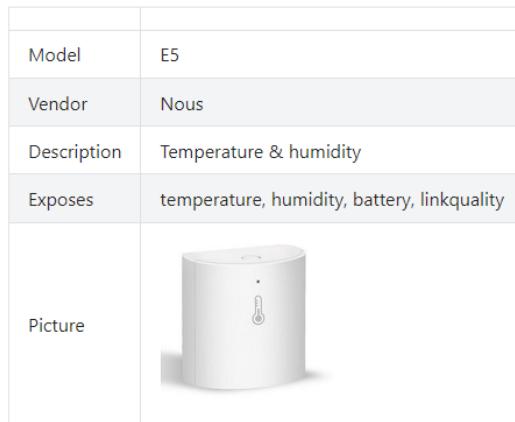


Figure 86 : Capteur de température et d'humidité

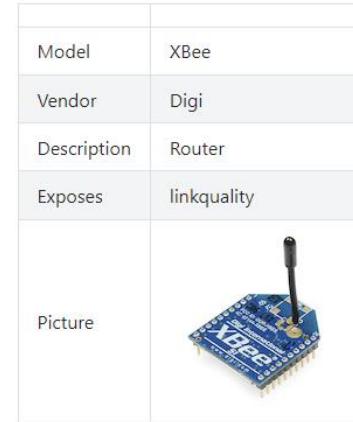


Figure 85 : XBee

En Comparaison avec un périphérique Zigbee que j'ai choisi aléatoirement (représenté ci-dessus à gauche) (capteur de température & d'humidité), le XBee n'expose aucune donnée utile à part la qualité de sa connexion Zigbee.

Je m'attendais personnellement à ce que l'on détecte les niveaux des pin I/O et éventuellement les trames reçues et envoyées en UART. Mais ce n'est pas le cas.

Cela étant dit, si nous disposons d'un de ces milliers de périphériques supportés par Zigbee2MQTT, toute l'application que j'ai effectuée fonctionnerait. Nous y verrons les données utiles, tel la température, l'humidité, la batterie et la qualité radio Zigbee de ce capteur.

Malgré tout, j'ai quand même essayé de finaliser cette approche avec quelques tests avec le Node Red :

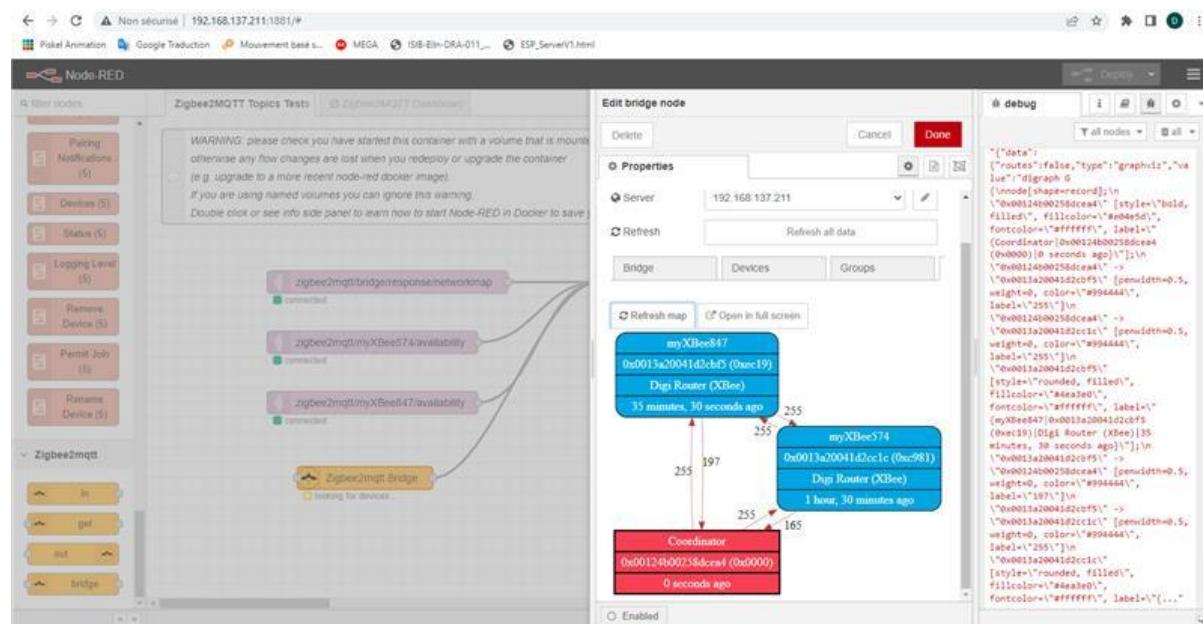


Figure 87 : Lecture des XBee sur NodeRed

Comme je m'y attendais, juste ces informations présentées dans le schéma ci-dessus sont disponibles. La solution est de créer un autre type de TP.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.3.6. Explication de la solution du TP Industrie 4.0 : TP Multi-Protocoles**

Pour rappel, le TP Industrie 4.0 est celui qui permettra aux étudiants de toucher à la majorité des aspects software réalisables avec l'ensemble des PCBs réunis. Le voici :

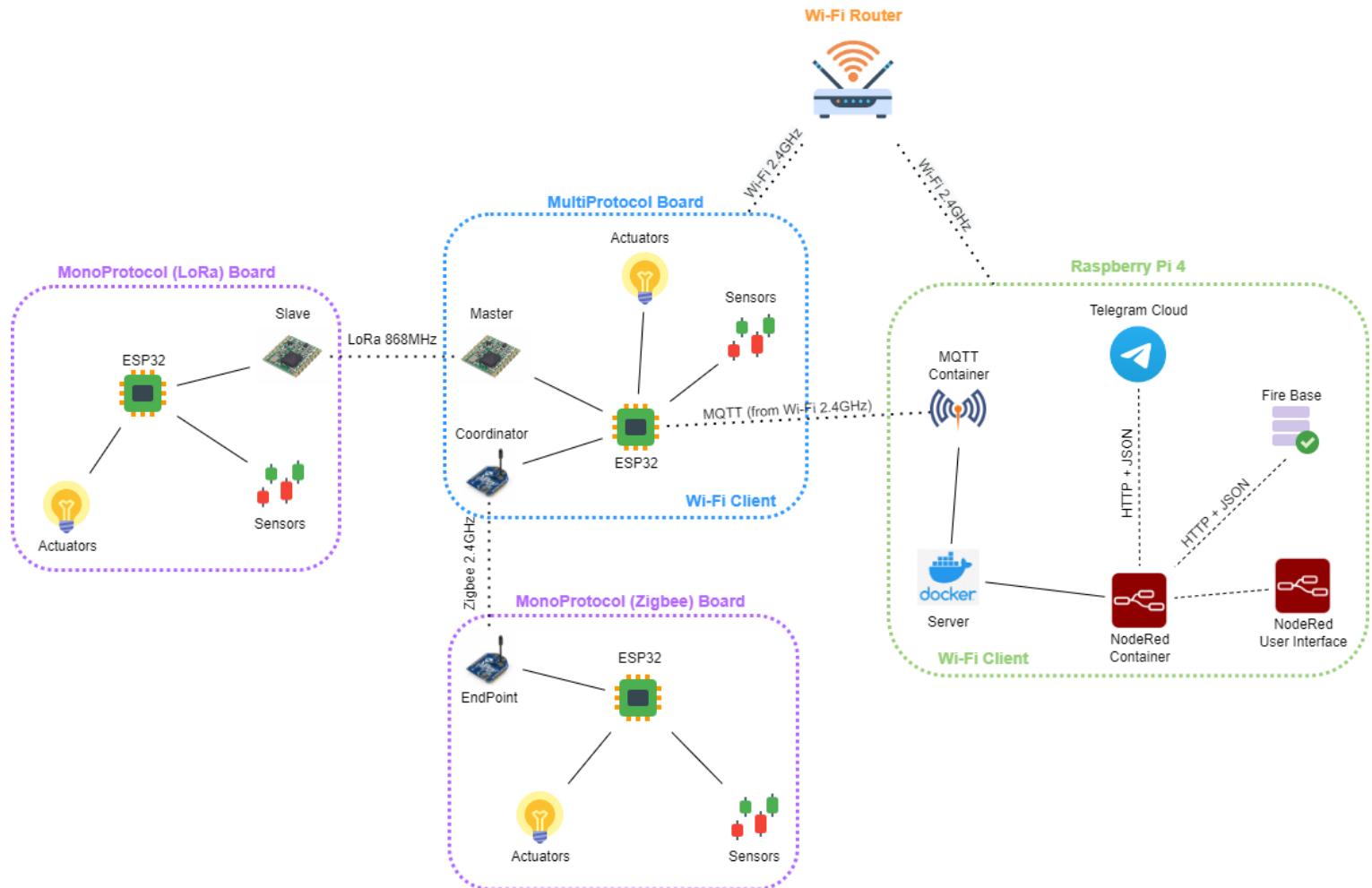


Figure 88 : TP Industry 4.0 - TP Multi-Protocole

Dans ce TP, j'utilise un Raspberry Pi 4, mais les étudiants pourraient très bien réussir à effectuer les mêmes étapes que m'apprête à présenter aux pages suivantes avec un PC.

3.3.6.1. Configuration initiale du Raspberry Pi 4

Cette étape n'est pas utile dans le cas où vous préférez utiliser un PC plutôt qu'un Raspberry Pi.

Lorsque l'on reçoit un Raspberry Pi 4, la première étape à réaliser avant toute chose est celle d'installer un OS.

Il y a plusieurs façons d'effectuer ceci, je me suis personnellement laissé guidé par un tutoriel (<https://www.youtube.com/watch?v=g9u6KleX7iU>) suivant plusieurs approches de configuration.

En premier temps, il faut télécharger le logiciel « Raspberry Pi Imager » pour installer le système d'exploitation. Plusieurs versions d'OS sont proposées. L'idéal dans notre type d'application serait d'installer celle du « Raspberry Pi OS Lite ». J'ai personnellement utilisé la version « Rasberry Pi OS with desktop and recommended software » car j'aimerais l'utiliser avec un écran dans l'avenir.

Ce logiciel demandera ensuite de choisir la carte SD correspondante au RaspberryPi. Si nous souhaitons avoir accès à d'autres paramètres, comme celui permettant l'accès au Wi-Fi local, il faut cliquer sur CTRL + MAJ + X. Cela ouvrira une nouvelle fenêtre du logiciel. Dans ces paramètres avancés, il faut cocher la case d'activation SSH. Ensuite insérer un nouveau mot de passe d'accès en ssh. Après introduire les paramètres du routeur Wi-Fi pour avoir une connexion instantanée au démarrage du Raspberry Pi.

Ensuite pour graver l'OS sur la carte SD du Raspberry Pi depuis notre PC, il faut appuyer sur Write dans le logiciel. Cela prendra un peu temps avec le rechargement.

Après cela, on retire la carte SD du PC pour ensuite la mettre dans son emplacement sur le Raspberry Pi. Il faudra l'alimenter et ensuite trouver son adresse IP. Plusieurs façons permettent cela. J'ai personnellement créé un point d'accès Wi-Fi depuis mon PC portable. Dans les paramètres du réseau de mon PC portable, le Raspberry est apparu connecté, sous forme de liste. Dans cette dernière se trouve son adresse IP.

Pour entrer en communication à distance en ssh avec le Raspberry pi, on ouvre l'invite de commande avec n'importe quel PC. Il faut que ce dernier soit connecté localement au même réseau sur lequel le Raspberry Pi est connecté. Dans mon cas, il s'agissait de mon PC portable, étant lui-même le « routeur ».

Ensuite, on insère cette commande :

```
C:\Users\David>ssh pi@192.168.137.211
```

Le mot de passe du raspberry pi sera demandé. Il s'agit de celui qu'on a choisi dans les paramètres avancés. Sinon, par défaut, le mot de passe est « raspberry ».

```
pi@raspberrypi : ~$ curl -sSL | sh
```

Pour terminer, on met à jour tous les dépôts de paquets en seulement deux lignes de commande :

```
pi@raspberrypi : ~$ sudo apt update
...
pi@raspberrypi : ~$ sudo apt upgrade
...
```

3.3.6.2. Implémentation du Docker

L'installation du docker est primordiale pour l'utilisation de l'application que j'ai effectuée. Là se trouveront les conteneurs des différents plugins (MQTT, Node Red & Portainer).

En théorie, on pourrait s'en passer, et installer nativement ces plugins à la racine du Raspberry Pi. Mais l'inconvénient en faisant ça, c'est que les configurations répétitives que nous apportons se font tous en ssh. Cela apporte un certain inconfort en comparaison avec les interfaces utilisateurs que propose Docker.

Le docker apporte non seulement des interfaces utilisateurs mais également d'autres outils pour gérer au mieux notre serveur. Je fais références aux sécurités proposés, aux API proposés, aux manipulations aisées des différents conteneurs (restart, off, on, logs, ...), etc...

Pour placer un docker dans le raspberry Pi 4 que je possède, j'ai eu recours à un autre tutoriel (<https://www.youtube.com/watch?v=rIVPCy3IU48>).

Je vais décrire les étapes que j'ai supposées essentiel à mon application que j'ai suivie tout au long de la vidéo.

Avant toute chose, le bon réflexe à avoir, est de mettre à jour la liste de nos paquets :

```
pi@raspberrypi : ~$ sudo apt update  
...  
pi@raspberrypi : ~$ sudo apt upgrade  
...
```

Ensuite, pour installer le docker, il y a cette commande à entrer :

```
pi@raspberrypi : ~$ curl -sSL https://get.docker.com | sh  
...
```

On va ensuite devoir rajouter notre utilisateur au groupe docker. Dans cet exemple, le nom d'utilisateur est « pi » :

```
pi@raspberrypi : ~$ sudo usermod -aG docker pi
```

A ce stade, le docker est installé.

3.3.6.3. Container Portainer

Portainer est une interface utilisateur permettant à l'utilisateur d'accéder à distance à la gestion entière du docker depuis l'adresse IP suivie du port 9000 de la machine virtuelle.

Si le docker est installé sur Windows, l'idéal serait d'avoir directement accès au docker depuis docker desktop. Il s'agit d'une interface utilisateur également, mais sous forme de programme téléchargé localement sur le PC.

Si le docker est installé sur Raspberry Pi, et que vous ne possédez pas d'écran pour visualiser docker desktop, la solution est d'installer portainer à la place. Ainsi, à partir de n'importe quel PC branché localement au même routeur wi-fi, on aura accès aux commandes docker depuis portainer en entrant l'adresse IP suivie du port dans le navigateur.

Voici la commande à entrer pour installer le portainer:

```
pi@raspberrypi : ~$ sudo docker pull portainer/portainer:linux-arm
...
```

Le dernier paramètre correspond au « <nom de l'éditeur>/<le nom du projet>:linux-<type de processeur> ».

Le « arm » est le type de processeur sur le Raspberry, mais aussi les GSM et autre. Si on est sur PC, on supprimera cet « arm ».

A ce stade, l'image a été récupérée. Maintenant on va créer un conteneur pour lancer et faire cette image :

```
pi@raspberrypi : ~$ sudo docker run --restart always -d -p 9000:9000 -v
/var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer:linux-arm
```

Dans cette commande nous devons nous rassurer que le conteneur démarre dès que le docker s'exécute. A parti de là, nous choisirons le port souhaité. Je l'ai laissé à 9000 par défaut.

Normalement, à ce stade, notre container portainer devrait tourner. Pour y accéder, il suffit insérer l'adresse IP de notre machine virtuelle (du raspberry pi, ou du PC) suivie du port 9000. On utilise n'importe quel navigateur, je l'ai par exemple exécuté avec le navigateur Chrome. Voici un exemple :

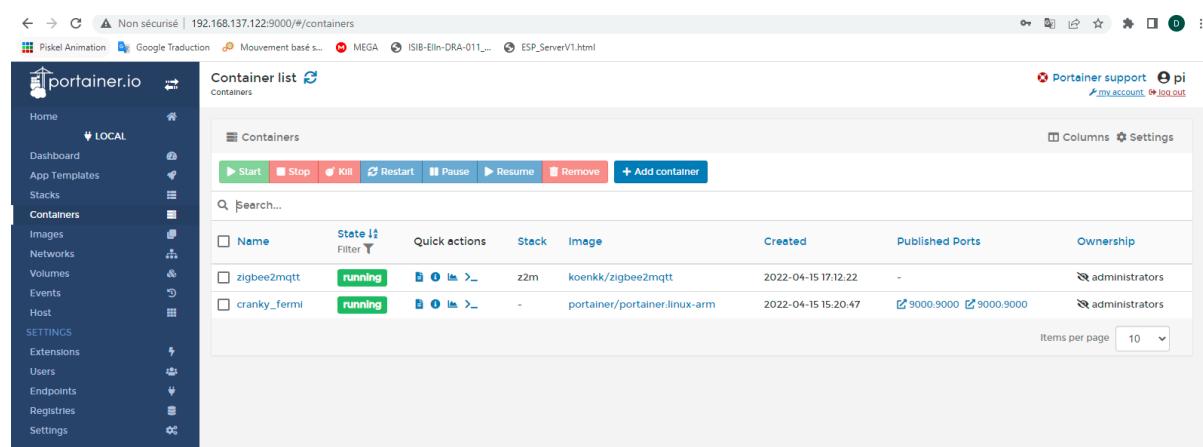


Figure 89 : Liste de containers sur Portainer

3.3.6.4. Container MQTT

Le courtier Mosquitto MQTT permet de connecter plusieurs nœuds connectés en Wi-Fi. Ces nœuds peuvent être des ESP32 par exemple. N'importe quel appareil IoT est compatible tant qu'il a une connexion Wi-Fi ou Ethernet.

MQTT est l'abréviation de Message Queuing Telemetry Transport. Le principe de messagerie que propose Mosquitto suit le système de publication et d'abonnement. Tous les appareils abonnés à un sujet spécifique reçoivent les messages que d'autres appareils publient sur ce même sujet :

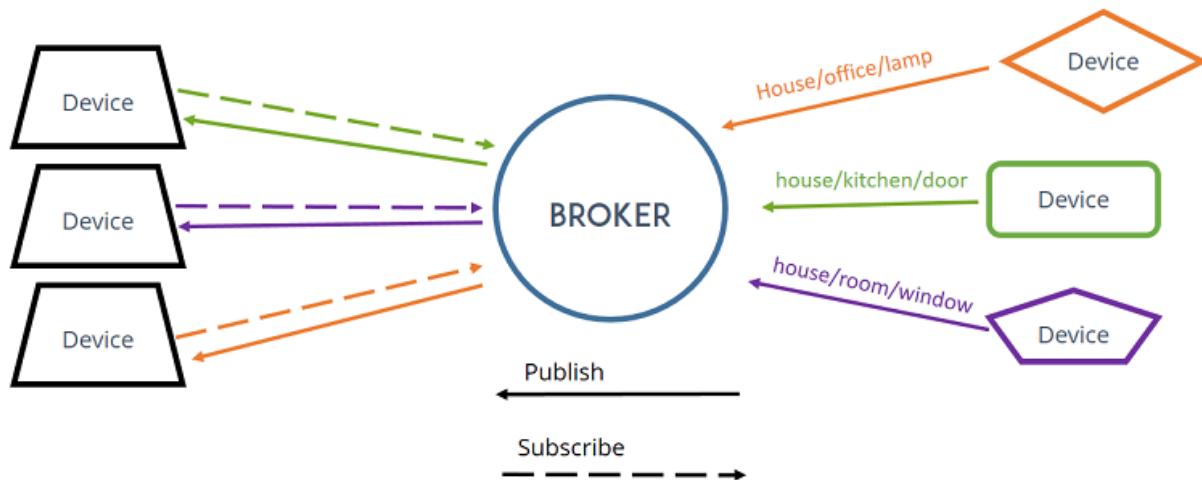


Figure 90 : MQTT processus

Au fur et à mesure, dans mes explications qui suivront, ce principe de MQTT sera de plus en plus détaillé.

J'ai personnellement installé le broker MQTT nativement dans le Raspberry Pi plutôt que le container Docker. La raison est que j'ai rencontré plusieurs problèmes que je ne saurais expliquer dans ce dossier.

En premier temps, nous exécutons le bon réflexe à avoir :

```
pi@raspberrypi : ~$ sudo apt update
...
pi@raspberrypi : ~$ sudo apt upgrade
...
```

Ainsi, nous sommes certains de démarrer notre installation MQTT sur des versions de paquets bien frais.

Pour installer le broker Mosquitto, il suffit d'entrer cette commande :

```
pi@raspberrypi : ~$ sudo apt install -y mosquitto mosquitto-clients
...
```

Ensuite, pour faire en sorte que le Mosquitto démarre automatiquement lorsque le Raspberry Pi s'allume, exécutez cette commande :

```
pi@raspberrypi : ~$ sudo systemctl enable mosquitto.service
...
```

Pour tester l'installation au niveau du volume :

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```
pi@raspberrypi : ~$ mosquito -v
...
```

Une réponse à cette exécution aura lieu. Il affichera des messages de la version actuellement téléchargée avec autres détails.

Pour accéder à distance, pour que nous puissions communiquer avec des autres périphériques IoT, il nous faudra modifier et créer un fichier de configuration. Nous utiliserons l'éditeur nano :

```
pi@raspberrypi : ~$ sudo nano /etc/mosquitto/mosquitto.conf
```

L'éditeur s'ouvrira. Le fichier aura quelques lignes déjà présentes, il nous suffira d'en insérer deux lignes de plus, celle correspondant au port, et celle autorisant l'accès au broker à n'importe qui :

```
pi@raspberrypi: ~
GNU nano 5.4          /etc/mosquitto/mosquitto.conf *
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

listener 1883
allow_anonymous true
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
 ^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^_ Go To Line

Figure 91 : Configuration Mosquitto

Etant donné que le broker Mosquitto est nativement installé dans le Raspberry Pi, pour exécuter ce nouveau broker, il faudra redémarrer le Raspberry :

```
pi@raspberrypi : ~$ sudo reboot
```

A ce stade, MQTT est prêt à l'emploi.

3.3.6.5. Container Node Red

Node Red agit comme un outil polyvalent permettant de connecter plusieurs périphériques, plusieurs API, et plusieurs services en ligne. Il est très utilisé dans le monde de l'IoT. Cet outil est basé sur le développement de programmation visuelle.

Dans le cadre du TP que je développe, NodeRed agit comme une passerelle permettant de guider le flux de plusieurs données vers des points de terminaison (Dashboard, Base de données, Cloud, API, ...).

Pour installer NodeRed sous forme de container Docker, voici la commande que j'ai utilisée :

```
pi@raspberrypi : ~$ docker run -it -p 1880:1880 -v node_red_data:/data --name mynodered
nodered/node-red
```

Et c'est tout. Ce container apparaîtra dans le portainer. Il nous suffira de le redémarrer depuis portainer. Ensuite, pour accéder à cet outil, il suffira d'entrer l'adresse IP suivie du port 1880 dans un navigateur.

3.3.6.6. API proposé par Telegram

Nous avons tous entendu parler de la messagerie Telegram comme WhatsApp, Messenger, Discord et plein d'autres réseaux sociaux. Telegram offre donc quelque chose de plus pour les grands passionnés de l'IoT. Il s'agit d'une API.

Une API est l'acronyme de « Application Programming Interface ». Il s'agit de solutions logicielles permettant à deux applications ou plus de communiquer entre elles. Plusieurs formats d'API existent, et celle proposée par Telegram est sous forme de requêtes HTTP.

L'application que j'ai effectuée avec Telegram suit plus une approche API que Cloud. Il reprend cependant quelques fonctionnalités d'un Cloud, sans trop aller plus loin.

Telegram n'est pas comparable aux autres Clouds à proprement parlé, mais intègre cependant un minimum de ces fonctionnalités Clouds. En effet, il détient une capacité de stockage. Par compte d'un utilisateur, ce cloud a une capacité atteignant jusqu'à 1.5Go gratuit. Ce qui le rend assez attractif.

Bref, il se trouve que cet API est disponible sous forme de bibliothèque dans NodeRed. Cette dernière est justement basée sur le principe de requêtes HTTP contenant des données utiles sous forme de JSON.

Mais avant, pour avoir accès à cet API, il nous est nécessaire de créer un compte Telegram. Comme WhatsApp, le compte est créé à partir du numéro de téléphone que vous possédez. Une fois que le compte est réalisé, une autre opération doit être effectuée, celle de créer un bot.

Un bot est une entité gérée et créée par un script pour interagir avec des messages d'un utilisateur. Il reprend le statut d'un ami par exemple, il a un nom, prénom et un avatar. Le bot peut répondre en fonction des interactions qu'il effectue avec l'utilisateur sur base du script qu'il embarque. Ce script sera créé et géré dans NodeRed dans mon cas.

Pour créer un bot, on cherche dans la barre de recherche de Telegram le bot « BotFather ». Ce dernier créera un bot pour vous. Il nous donnera des identifiants ainsi une clef permettant d'accéder en toute sécurité à des ressources depuis le script que nous créerons.

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Lorsque nous sommes en contact avec BotFather, on insère le message « /newbot ».

Il nous demandera quel nom nous souhaitons lui donner. Comme représenté ci-dessous, il créera le bot comme souhaité et nous donnera toutes les données relatives à ce nouveau bot.

Par sécurité, certains tutoriels cachent la clé permettant d'accéder à l'API HTTP. Dans mon cas, étant donné que son utilisation est entièrement dédiée à mon TFE, je ne le cache pas : 5365246871:AAHoyLAUHPowHowEMO0kNbou-NA4JZDXh4E.

Il s'agit en quelque sorte de la carte d'identité du nouveau bot. Il se nomme ISATtelegram. Sur cette clé se trouvent deux paramètres, le premier est le ChatID (5365246871), et le deuxième, suivi des deux petits points, est l'identifiant du bot.

A ce stade, toute interaction avec le bot est possible. Nous pourrons le trouver dans nos contacts, et pourrons aussi commencer à le programmer.

Comme je l'avais évoqué précédemment, la programmation de ce bot est réalisée sur NodeRed. Nous pouvons jeter un coup d'œil au code présenté dans l'annexe 30.

En premier temps, deux détails nous intéressent dans ce code : la connexion MQTT & Telegram.

A ce stade, Telegram est fraîchement introduit, je commencerai avec l'explication de son code. Ensuite, j'introduirai le processus MQTT établi.

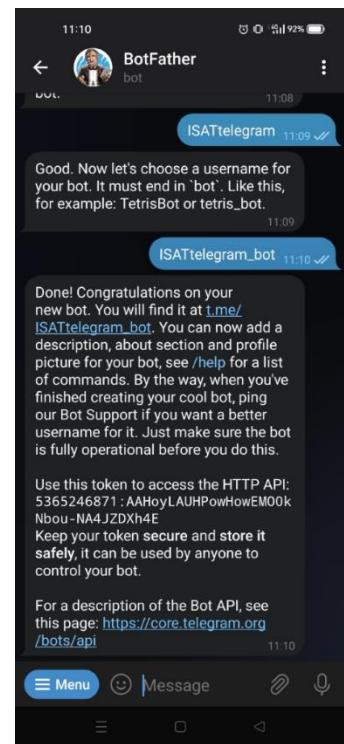


Figure 92 : BotFather sur Telegram

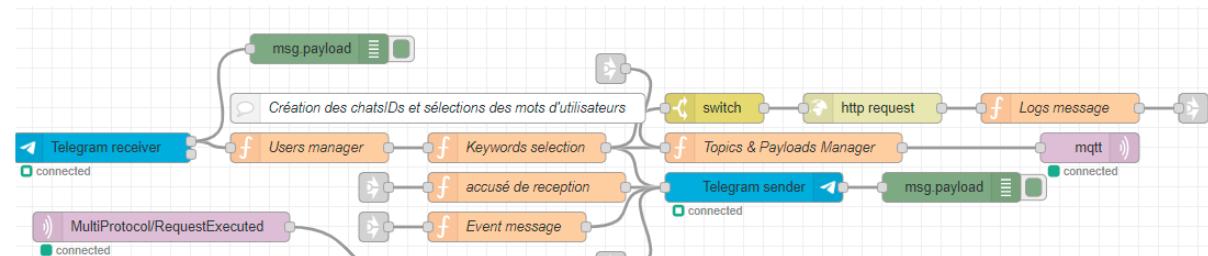


Figure 93 : Gestion du bot Telegram sur NodeRed

Dans le premier bloc bleu présenté ci-dessus à gauche se trouve un premier nœud : le récepteur de message Telegram. Là nous y configurons le bot auquel le nœud sera lié à partir des identifiants précédemment reçus par BotFather. Peu importe la personne qui essaie de communiquer avec le bot sur Telegram, ce nœud le détecte. Il récupère les informations du message entrant. Ces informations sont stockées sous forme JSON. On y retrouve le nom d'utilisateur, son ChatID, son message, etc...

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

3.3.6.7. Interactions entre l'utilisateur Telegram, les PCBs et le bot

En se basant sur ces informations JSON, le nœud suivant (Users manager) agira sur base du code que je le lui ai implémenté (voir annexe 32). Ce script filtre les utilisateurs autorisés pour savoir s'il doit lui répondre par la suite ou pas.

Le nœud suivant (voir annexe 33) crée une interface interactive entre l'utilisateur et le bot. Cette interface est composée d'un clavier personnalisé.

Dans cette fonction, le script se base sur un dictionnaire que j'ai créé pour placer des boutons des claviers comme paramétrés. Ensuite, une boucle ira lire les clefs & valeurs de façon à générer pas à pas les boutons sur base des positions souhaitées. Le bot sait donc quoi faire, alors il répondra à l'utilisateur le résultat visuel suivant :

Ceci a été possible grâce au nœud « Telegram Sender » présenté en bleu dans le programme du NodeRed. Celui-ci a assemblé le JSON correspondant au clavier personnalisé et l'a envoyé en http vers Telegram

Ce clavier est un raccourci pour l'utilisateur. À la place de créer manuellement les messages au bot, l'utilisateur clique sur un bouton et le message correspondant s'envoie automatiquement. C'est sur ce principe que l'application Telegram de mon TFE est basée.

Lorsqu'un message est envoyé, il y a le nœud qui décide s'il s'agit d'une requête MQTT à envoyer (voir annexe 34). Ce nœud va prendre l'identifiant du bouton pour créer une demande sous le Topic correspondant.

Un Topic est un sujet auquel un message MQTT s'abonne. Par exemple, dans le cas où nous suivons l'image ci-dessus :

L'utilisateur est dans les paramètres du PCB MultiProtocol et il clique sur « Get DHT22 Temperature ». Il y a alors création du Topic « MultiProtocol/Request » avec un message contenant « temp ». En d'autres termes, il demande à l'ESP32, qui est dans le PCB MultiProtocol, de lui envoyer la température.

Le code de l'ESP32 du PCB MultiProtocol (voir annexe 35) récupère ensuite la température du DHT22 avant de le publier sous le topic « MultiProtocol/RequestExecuted ». Cette requête exécutée est détectée dans le nœud NodeRed présenté en rose à gauche (voir annexe 30).

Lorsque depuis NodeRed la requête exécutée a été détectée, la donnée utile (payload) s'achemine vers le nœud « accusé de réception ». Ce nœud formate un message de façon à ce que la donnée soit présentable lorsqu'elle est envoyée à l'utilisateur sur Telegram (comme représenté dans l'image ci-dessus).

Lorsque l'utilisateur ne choisit pas de récupérer la température du PCB MultiProtocole, mais plutôt d'actionner le buzzer du PCB orienté Zigbee, le principe est le même, à un détail prêt.

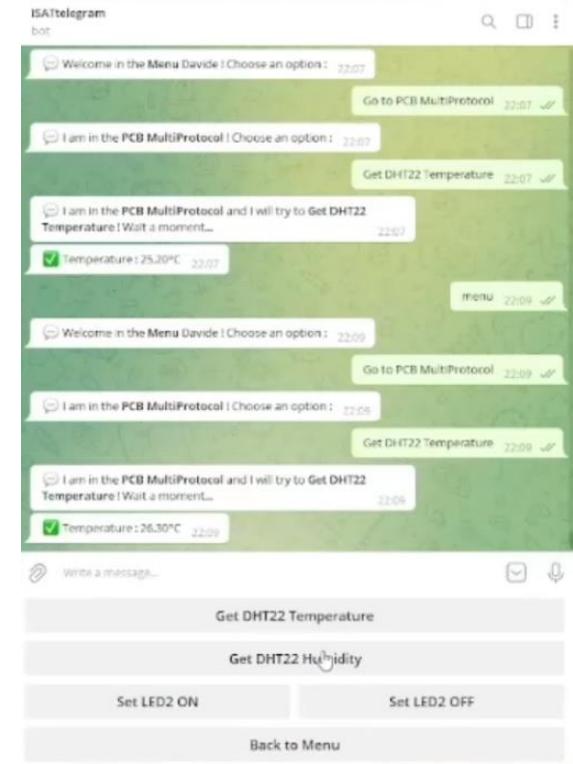


Figure 94 : Interface utilisateur Telegram

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Voici le Topic MQTT correspondant à l'appui du bouton « Set Buzzer ON » : « MonoProtocol Zigbee/Request » avec le message « Bon » ('B' pour buzzer et 'on' pour ON).

L'ESP32 qui est abonné à ce Topic est celui du PCB MultiProtocol. Il est d'ailleurs abonné à tous les Topic.

Ce PCB ira demander à son coordinateur Zigbee (premier module Zigbee) d'envoyer un signal radio. Le point de terminaison du PCB MonoProtocol orienté Zigbee (donc le deuxième module Zigbee) recevra la demande par radio et allumera le buzzer (code ESP32 ; voir annexe 36).

Afin que l'utilisateur soit certain que le buzzer soit allumé (même si on l'entend), un signal d'accusé de réception prend le chemin inverse. Il apparaîtra sous le topic « MonoProtocol Zigbee/RequestExecuted » dans NodeRed depuis le PCB MultiProtocol. A nouveau, le nœud « accusé de réception » formatera le contenu du message reçu avant de l'afficher dans le NodeRed.

En ce qui concerne l'interaction avec le PCB LoRa, il s'agit du même processus qu'avec le PCB Zigbee. Donc tout se passe en premier temps du côté du PCB MultiProtocol avant d'aller à la destination voulue (PCB Zigbee, PCB LoRa, ou lui-même : PCB MultiProtocol).

Bref, on remarque très vite qu'avec ce type d'application, il est possible d'aller très loin.

En effet, je ne me suis pas arrêté là, j'ai également développé une gestion d'évènement. Cela signifie que si un utilisateur appuie sur un des boutons poussoirs présents sur un des 3 PCBs, une notification aura lieu comme représenté ci-contre.

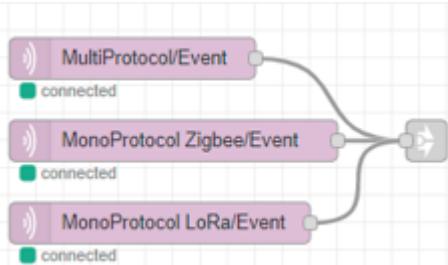


Figure 96 : Topics MQTT des événements

Ce principe d'évènement par notification suit plus ou moins le même principe qu'est expliqué ci-dessus. La différence est que le Topic avec lequel l'événement a lieu n'est pas identique.

En effet, comme représenté ci-contre, les Topic sont terminés par le mot « .../Event ». Leur payload sera acheminé vers un nœud qui formatera la donnée avant de l'envoyer sur Telegram.



Figure 95 : Interface utilisateur Telegram



Figure 97 : Notification Telegram

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**3.3.6.8. API proposé par Fire Base**

Fire Base propose des services de toute sorte, notamment celles d'hébergement et de base de données. Je vais m'attarder plus spécifiquement sur le principe de base de données que propose ce dernier, et plus précisément sur celle en temps réel.

Dans l'annexe 38 nous avons un aperçu de cette base de données. Là, se trouvent des données, et pas n'importe lesquelles.

En effet, à chaque requête exécutée depuis Telegram, la donnée utile reçue de la part d'un PCB est stockée sous son dernier état. De plus, pour garder une trace, un système de logs est également présent dans mes données.

A nouveau, la gestion de l'écriture vers la base de données est faite sur NodRed. Etant donné que Firebase propose également une API, je me suis permis d'en profiter pour voir comment elle est gérée. Voici le script me permettant d'effectuer toutes mes actions sur la base de données.



Figure 98 : Gestion des données Firebase sur NodeRed

Analysons la première ligne, lorsqu'un évènement a eu lieu (un appui sur un bouton poussoir d'un des 3 PCBs), le message que contient le Topic sur des événements correspondants est acheminé vers le nœud supérieur « simpletime ». Ce dernier récupère le temps (année : mois : jour : heure : minute : seconde) depuis un serveur ntp. Ensuite, le nœud « to Event Sections Firebase – display payload » se charge de formater la donnée utile (“BP1 appuyé sur le PCB Multiprotocol”) en le concaténant avec le temps précédemment acquis. Enfin, toujours dans ce même nœud, il y a la construction de la requête API HTTP avec le message JSON qu'il intègre :

```
recv = msg.payload;
url = "https://isatfirebase-default-rtdb.firebaseio.com/app/";

msg.method = "PUT";
msg.url = url+"Event.json";

msg.payload = JSON.stringify(msg.myymd + " " + msg.mytimes + ":" + recv);

return msg;
```

Voici une illustration :

le JSON aurait donc comme paramètres (clé, valeur) :

- msg.method = “PUT”
- msg.url = “<https://isatfirebase-default-rtdb.firebaseio.com/app/Event.json>”
- msg.payload = “ 2022-05-18 12:49:41 : *BP2* in the *PCB MonoProtocol LoRa* is pushed !”

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Ce JSON sera ensuite envoyés en HTTP par le biais du nœud présent à l'extrême droite « http request ». Fire Base détectera qu'une commande de type PUT est arrivée. Il écrasera donc l'ancienne valeur précédemment présente par cette nouvelle valeur.

En ce qui concerne la deuxième ligne représentée dans le programme NodeRed à la page précédente, le même principe est utilisé. Il s'agit de la ligne sur laquelle nous stockons les données des différents capteurs et actionneurs des différents PCBs dans le FireBase. La différence n'est située qu'à la destination de la donnée utile à stocker. L'un est stocké dans l'onglet “.../MultiProtocol/temp.json”, l'autre dans “.../MonoProtocol Zigbee/Bon.json” etc...

Les trois dernières lignes des nœuds du programme NodeRed correspondent à la gestion des Logs. Là se trouvent deux types de commande HTTP : PUT & DELETE. Afin de limiter un maximum de ligne, j'ai défini une limite : 15 lignes maximum. A chaque fois qu'une nouvelle ligne s'ajoute, la ligne ayant la position la plus éloignée (15 lignes plus loin, la ligne la plus ancienne) se supprime.

Lorsqu'il y a écriture et suppression, il nous est possible de voir en temps réel les actions effectuées par FireBase. On peut observer qu'une ligne verte corresponde à une ligne ajoutée, et qu'une ligne rouge corresponde à une ligne supprimée. Tout est assez direct, c'est fascinant de voir cela en temps réel.

Les délais placés dans le programme NodeRed sont utiles dans le cas où plusieurs requêtes HTTP sont demandées en même temps. Pour éviter tout chevauchement, toute collision éventuelle, ces délais cadencent l'ordre du passage à l'unité.

3.3.6.9. Interface utilisateur du Node Red

La réalisation d'une interface utilisateur est également disponible depuis NodeRed. La programmation de celle-ci que j'ai effectuée est présente dans l'annexes 31.

En défiant, des graphiques sont également présents. Ils récupèrent les données utiles et les places sous

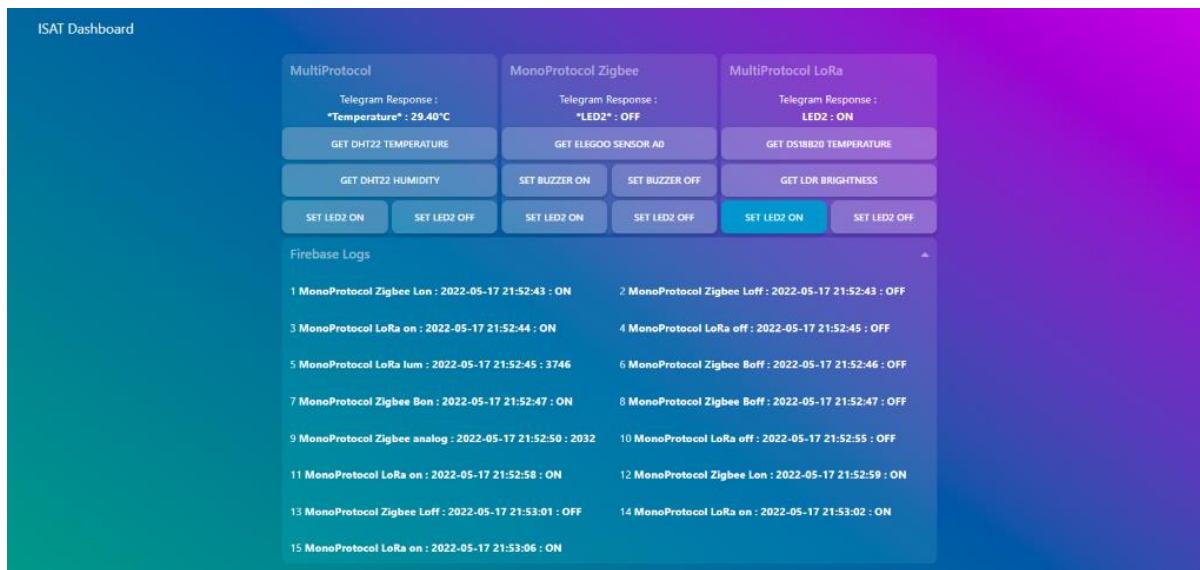


Figure 99 : Interface utilisateur NodeRed

forme de courbe en fonction du temps. Le principe de fonctionnement de cette interface est identique à celui de Telegram. Je fais référence aux différents menus et aux boutons qu'ils intègrent principalement. Je ne passe par aucune API puisque j'ai accès en direct localement à tous les flux de données présents dans NodeRed.

4. Conclusion

J'ai essayé d'assembler un maximum d'éléments évoqués dans le cahier des charges dans mon projet. Les PCBs ont été testés, et sont aujourd'hui capables d'exécuter toutes les tâches citées dans ce rapport. Malheureusement, j'ai perdu un précieux temps lors du développement de l'approche Zigbee2MQTT. En effet, après 3 semaines de développement, j'ai rencontré une limite de ressources m'empêchant d'aller plus loin. En conséquence, l'aspect « Cloud » demandé dans le cahier des charges ne m'a pas pu être réalisé. J'ai nonobstant essayé de trouver une solution simple (Arduino IoT Cloud) pour gagner du temps, mais le résultat ne m'a pas convaincu. Les services proposés étaient trop orientés pour des utilisations d'amateurs et non professionnelles.

J'ai longtemps pensé à une V3.0 du PCB Multi-Protocole. Actuellement, en termes de taille, ce circuit imprimé mesure 100x106mm. L'idée serait de l'augmenter jusqu'au format euro (100x160mm) et de lui rajouter encore plus de périphériques : un buzzer, un driver de moteur pas à pas, un afficheur 7 segments, etc... Mais le plus intéressant serait de lui instaurer un connecteur RJ45 Ethernet. Cela apporterait au PCB une nouvelle couche physique de communication sur lequel il pourrait s'appuyer dans de nouvelles aventures.

Bref, comme on peut le constater, à ce jour, le monde est en constante évolution. Ce qui tire les ficelles n'est rien d'autre que la technologie. Fort heureusement, la nature humaine fait de pair avec l'évolution. Cependant, s'adapter en tant qu'utilisateur est une chose, mais dompter cette technologie en est une autre. D'où le fait que les métiers naissent, tandis que d'autres s'adaptent. Et il se trouve que le domaine dans lequel mon TFE s'oriente est directement en lien avec cette technologie colossale. Que l'on soit débutant, amateur ou étudiant dans ces nouveaux métiers, et tant qu'il y a une part de motivation, il nous est possible de suivre les avancées technologiques malgré leur complexité. Il faut se dire que même en période de pénurie des composants électroniques, les passionnés que je suis auront toujours une solution pour apprendre plus.

C'est sur ces paroles que j'invite les futurs détenteurs des PCBs à découvrir ce nouveau monde. Je pense que ce n'est qu'avec ce genre de matériel didactique que l'on peut bien apprendre.

Je suis satisfait du projet que j'ai mené tout au long de mon stage. Les différentes approches que j'ai réalisées m'ont permis d'acquérir un certain niveau de maturité dans le domaine. Avec un maître électronicien à mes côtés, j'ai pu développer et surpasser mes anciennes méthodes de travail. Après tous les efforts que j'ai effectués dans ce projet, je ne me suis malgré tout pas encore saturé. C'est pourquoi je pense être capable de poursuivre mes études dans ce domaine. Certains aspects ne me font plus peur, je me sens plus confiant. Je voyais initialement ce stage comme une préparation à la poursuite de mes études, et je ne suis pas déçu, loin de là.

Bibliographie

Documentations techniques et figures:

Electronicshub, getting started with esp32,
<https://www.electronicshub.org/getting-started-with-esp32/>,
Consulté le 20 02 22

Framboise314, raspberry pi 4 4 nouveautés qui vont vous faire craquer,
<https://www.framboise314.fr/raspberry-pi-4-4-nouveautes-qui-vont-vous-faire-craquer/#:~:text=Le%20Raspberry%20Pi%204%20Mod%C3%A8le,Cortex%20A53%20%3D%3E%20Cortex%20A72>,
Consulté le 17 04 22

Futura sciences, Internet Wi-Fi,
<https://www.futura-sciences.com/tech/definitions/internet-wi-fi-1648/>,
Consulté le 09 03 22

Bluetootth, learn about Bluetooth,
<https://www.bluetooth.com/learn-about-bluetooth/tech-overview/>,
Consulté le 09 03 22

Randomnerdtutorial, ESP32 Bluetooth low energy ble Arduino ide,
<https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/>,
Consulté le 09 03 22

Digi, Xbee VS Zigbee a simple comparaison,
<https://www.digi.com/blog/post/xbee-vs-zigbee-a-simple-comparison-guide>,
Consulté le 09 03 22

Rfwireless world, LoRa technology basics,
<https://www.rfwireless-world.com/Terminology/LoRa-technology-basics.html#:~:text=LoRa%20stands%20for%20Long%20Range%20Radio.%20It%20is,connect%20multiple%20applications%20running%20in%20the%20same%20network>,
Consulté le 27 05 22

Mokolora, what is lorawan,
<https://www.mokolora.com/fr/what-is-lorawan/>,
Consulté le 27 05 22

Emnify, lorawan,
<https://www.emnify.com/iot-glossary/lorawan>,
Consulté le 27 05 22

havr, Sigfox,
<https://www.havr.io/sigfox>,
Consulté le 29 05 22

condexatenbay, protocols infrarouges,
<https://www.condexatedenbay.com/protocoles-infrarouges/>,
Consulté le 29 05 22

thegeekpub, how I²C works I²C explained,
<https://www.thegeekpub.com/18351/how-i2c-works-i2c-explained-simply/>,
Consulté le 29 05 22

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

corelis, SPI tutorial,

<https://www.corelis.com/education/tutorials/spi-tutorial/#:~:text=SPI%20%28Serial%20Peripheral%20Interface%29%20is%20an%20interface%20bus,synchronous%20serial%20communication%20between%20master%20and%20slave%20devices>,

Consulté le 29 05 22

protosupplies, MCP2562 CAN bus transceiver,

<https://protosupplies.com/product/mcp2562-can-bus-transceiver/>,

Consulté le 29 05 22

Arduinomodules,

<https://arduinomodules.info/>,

Consulté le 8 04 22

Az delivery, esp32 developmentboard,

<https://www.az-delivery.de/fr/products/esp32-developmentboard>,

Consulté le 09 04 22

Image made in china, 45 in 1 sensors modules starter kit for Arduino,

<https://image.made-in-china.com/2f0j00IGFUIYPdnBkK/45-in-1-Sensors-Modules-Starter-Kit-for-Arduino.jpg>,

Consulté le 09 04 22

Myonlinestore, keyboard image

<https://cdn.myonlinestore.eu/e40d5160-bac0-4897-baae-d6065a5d5915/image/cache/full/a727adef9009ef4310c4ab69e0a53a4e3db5f1b1.jpg>,

Consulté le 09 04 22

Digikey, techniques for robust touch sensing design,

<https://www.digikey.com/en/articles/techniques-for-robust-touch-sensing-design>,

Consulté le 21 05 22

Howtomechatronics, DHT11 VS DHT22 specification parametters,

<https://howtomechatronics.com/wp-content/uploads/2016/01/DHT11-vs-DHT22-specifications-parameters.png?ezimgfmt=rs:352x225/rscb2/ng:webp/ngcb2>,

Consulté le 21 05 22

Elprocus, DS18B20 temperature sensor,

<https://www.elprocus.com/ds18b20-temperature-sensor/#:~:text=The%20DS18B20%20is%20one%20type,communicate%20with%20an%20inner%20microprocessor>,

Consulté le 21 05 22

Randomnumerdtutorials, how to install mosquito broker on raspberry pi,

<https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/>,

Consulté le 16 04 22

Nodered, docker,

<https://nodered.org/docs/getting-started/docker>,

Consulté le 16 04 22

Informatique mania, comment utiliser le cloud telegram pour stocker mes fichiers,

<https://www.informatique-mania.com/reseaux-sociaux/comment-utiliser-le-cloud-telegram-pour-stocker-mes-fichiers/>,

Consulté le 02 05 22

Table des figures :

Figure 1 : Raspberry Pi 4	15
Figure 2 : Raspberry Pi 4	15
Figure 3 : ESP32	16
Figure 4 : point à point (P2P)	17
Figure 5 : P2P, Diffusion & Maillage	17
Figure 6 : XBee SCHEMA	18
Figure 7 : LoRa SCHEMA	19
Figure 8 : LoRaWAN Processus	19
Figure 9 : SigFox Backend datas	20
Figure 10 : SigFox SCHEMA	20
Figure 11 : IR SCHEMA	21
Figure 12 : SPI Processus	21
Figure 13 : I ² C Processus	22
Figure 14 : CAN SCHEMA	23
Figure 15 : Potentiomètre	26
Figure 16 : Boutons poussoirs	26
Figure 17 : LEDs	26
Figure 18 : LEDs Neopixel	26
Figure 19 : LED infrarouge	26
Figure 20 : Récepteur infrarouge	27
Figure 21 : Module PIR	27
Figure 22 : Kit de capteurs d'Elegoo	27
Figure 23 : MPU 6050	28
Figure 24 : OLED	28
Figure 25 : MCP 23008	28
Figure 26 : Keyboard	28
Figure 27 : LDR	29
Figure 28 : DS18B20	29
Figure 29 : HC-SR04	29
Figure 30 : Buzzer	29
Figure 31 : Servomoteur	30
Figure 32 : DHT22	30
Figure 33 : RFID	30
Figure 34 : Click Boards of MikroBUS	31
Figure 35 : MikroBUS Pinouts	31
Figure 36 : TP Industry 4.0 - TP Multi-Protocole	33
Figure 37 : Schema bloc - PCB Multi-Protocole	35
Figure 38 : SCHEMA Jumpers - PCB Multi-Protocole	37
Figure 39 : SCHEMA Alimentation - PCB Multi-Protocole	37
Figure 40 : SCHEMA I ² C modules - PCB Multi-Protocole	38
Figure 41 : 3D - PCB Multi-Protocole	39
Figure 42 : Inkscape - PCB Multi-Protocole	39
Figure 43 : Ligne de commande Fusion360 - PCB Multi-Protocole	40
Figure 44 : Résultat SVG to PCB - PCB Multi-Protocole	40
Figure 45 : PCB touch button - PCB Mono-Protocole Zigbee	41
Figure 46 : Touch Button processus	41
Figure 47 : Schema bloc - PCB Mono-Protocole Zigbee	42

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Figure 48 : SCHEMA jumpers - PCB Mono-Protocole Zigbee	43
Figure 49 : SCHEMA MikroBUS - PCB Mono-Protocole Zigbee.....	44
Figure 50 : SCHEMA Buzzer - PCB Mono-Protocole Zigbee	44
Figure 51 : SCHEMA Bus Jumers - PCB Mono-Protocole Zigbee.....	44
Figure 52 : SCHEMA I ² C connecteur - PCB Mono-Protocole Zigbee.....	44
Figure 53 : 3D - PCB Mono-Protocole Zigbee	45
Figure 54 : Schema bloc - PCB Mono-Protocole LoRa.....	47
Figure 55 : SCHEMA LED - PCB Mono-Protocole LoRa.....	49
Figure 56 : SCHEMA DS18B20 - PCB Mono-Protocole LoRa.....	49
Figure 57 : 3D - PCB Mono-Protocole LoRa	50
Figure 58 : ROUTAGE Plan de cuivre - PCB Mono-Protocole LoRa	50
Figure 59 : Schema Bloc - PCB Mono-Protocole SigFox	52
Figure 60 : SCHEMA Jumpers - PCB Mono-Protocole SigFox.....	53
Figure 61 : Erreur de démarrage ESP32 - PCB Mono-Protocole SigFox	54
Figure 62 : SCHEMA condensateur EN - PCB Mono-Protocole SigFox.....	54
Figure 63 : SCHEMA LEDs et Boutons - PCB Mono-Protocole SigFox	54
Figure 64 : 3D - PCB Mono-Protocole SigFox.....	55
Figure 65 : Strapping Pins.....	57
Figure 66 : Pontage du PCB Mono-Protocole Zigbee	58
Figure 67 : Trou métalisé mal placé dans le PCB Mono-Protocole LoRa.....	59
Figure 68 : Coupure d'un signal dans le PCB Mono-Protocole LoRa	60
Figure 69 : Pontage du PCB Mono-Protocole LoRa.....	60
Figure 70 : Support MPU6050.....	61
Figure 71 : Support OLED (4 trous)	61
Figure 72 : Support OLED (2 trous)	61
Figure 73 : Encombrement de l'OLED dans le PCB Multi-Protocole	61
Figure 74 : Support PIR	61
Figure 75 : Dongle Zigbee2MQTT	62
Figure 76 : docker-compose Zigbee2MQTT	63
Figure 77 : configuration Zigbee2MQTT	64
Figure 78 : LOGS Dongle non reconnu	65
Figure 79 : XBee non reconnu	66
Figure 80 : LOGS XBee non reconnu par le dongle.....	66
Figure 81 : Clé de chiffrement XCTU	67
Figure 82 : LOGS XBee reconnu par le dongle.....	67
Figure 83 : Liste des XBee reconnus	67
Figure 84 : Premier maillage Zigbee.....	67
Figure 85 : XBee	68
Figure 86 : Capteur de température et d'humidité	68
Figure 87 : Lecture des XBee sur NodeRed	68
Figure 88 : TP Industry 4.0 - TP Multi-Protocole.....	69
Figure 89 : Liste de containers sur Portainer	72
Figure 90 : MQTT processus	73
Figure 91 : Configuration Mosquitto	74
Figure 92 : BotFather sur Telegram	76
Figure 93 : Gestion du bot Telegram sur NodeRed	76
Figure 94 : Interface utilisateur Telegram.....	77
Figure 95 : Interface utilisateur Telegram.....	78
Figure 96 : Topics MQTT des événements.....	78
Figure 97 : Notification Telegram	78

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

Figure 98 : Gestion des données FireBase sur NodeRed	79
Figure 99 : Interface utilisateur NodeRed.....	80

Annexes**■ Annexe 1 : Restrictions des pinouts de l'ESP32**

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fail if pulled high
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	

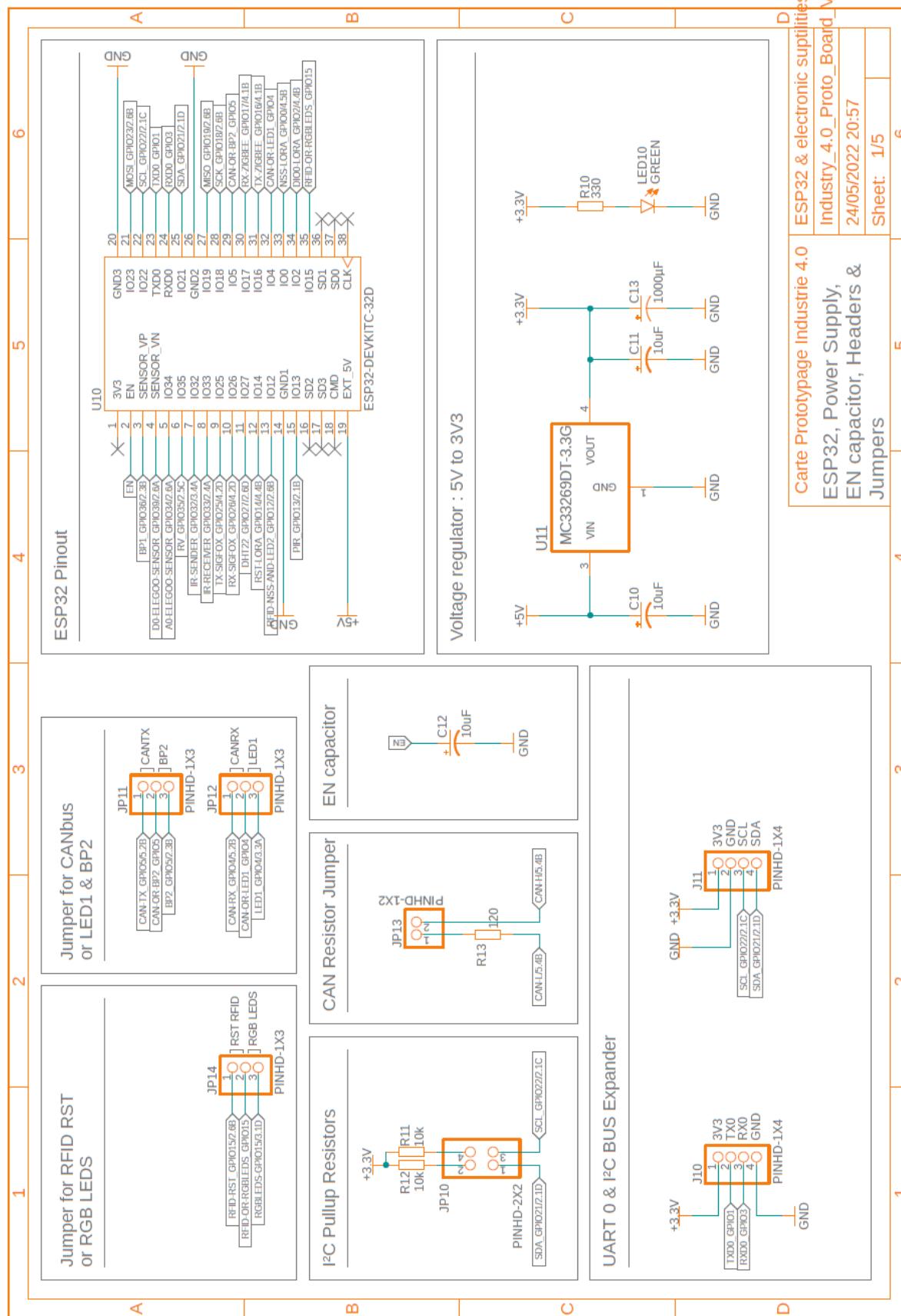
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

Sources : <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>

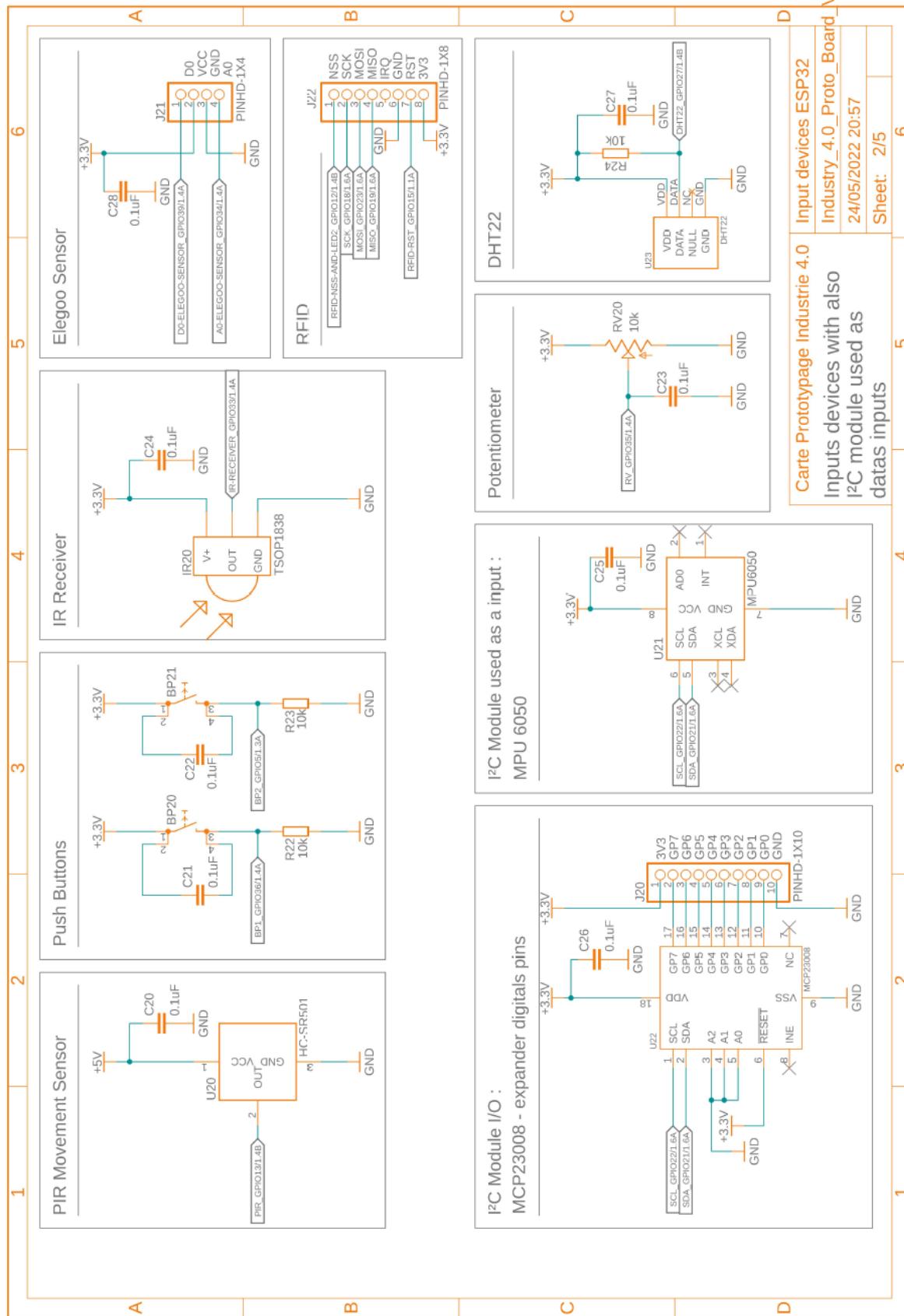
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 2 : PCB Multi-Protocole : SCHEMA page 1/5



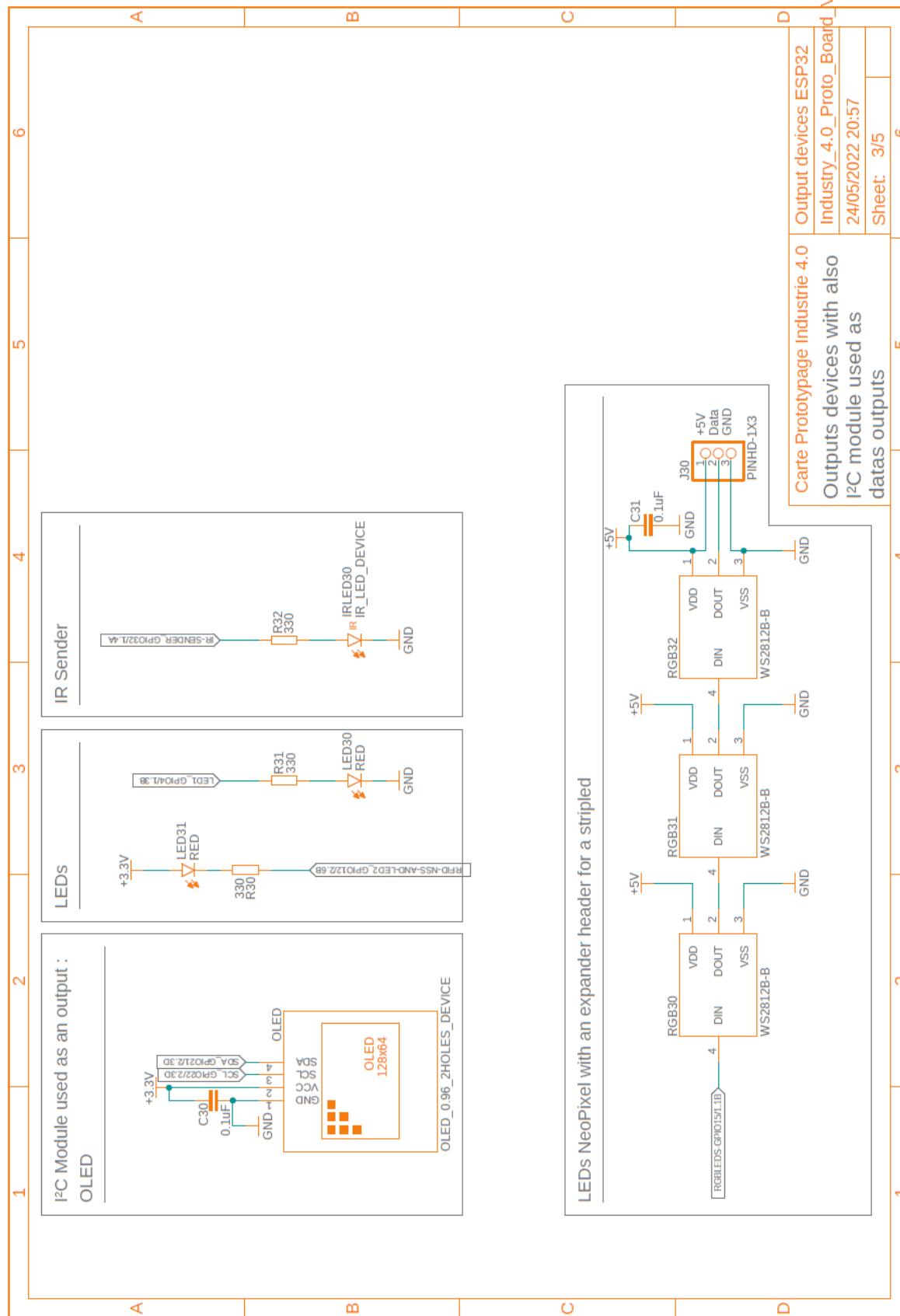
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 3 : PCB Multi-Protocole : SCHEMA page 2/5



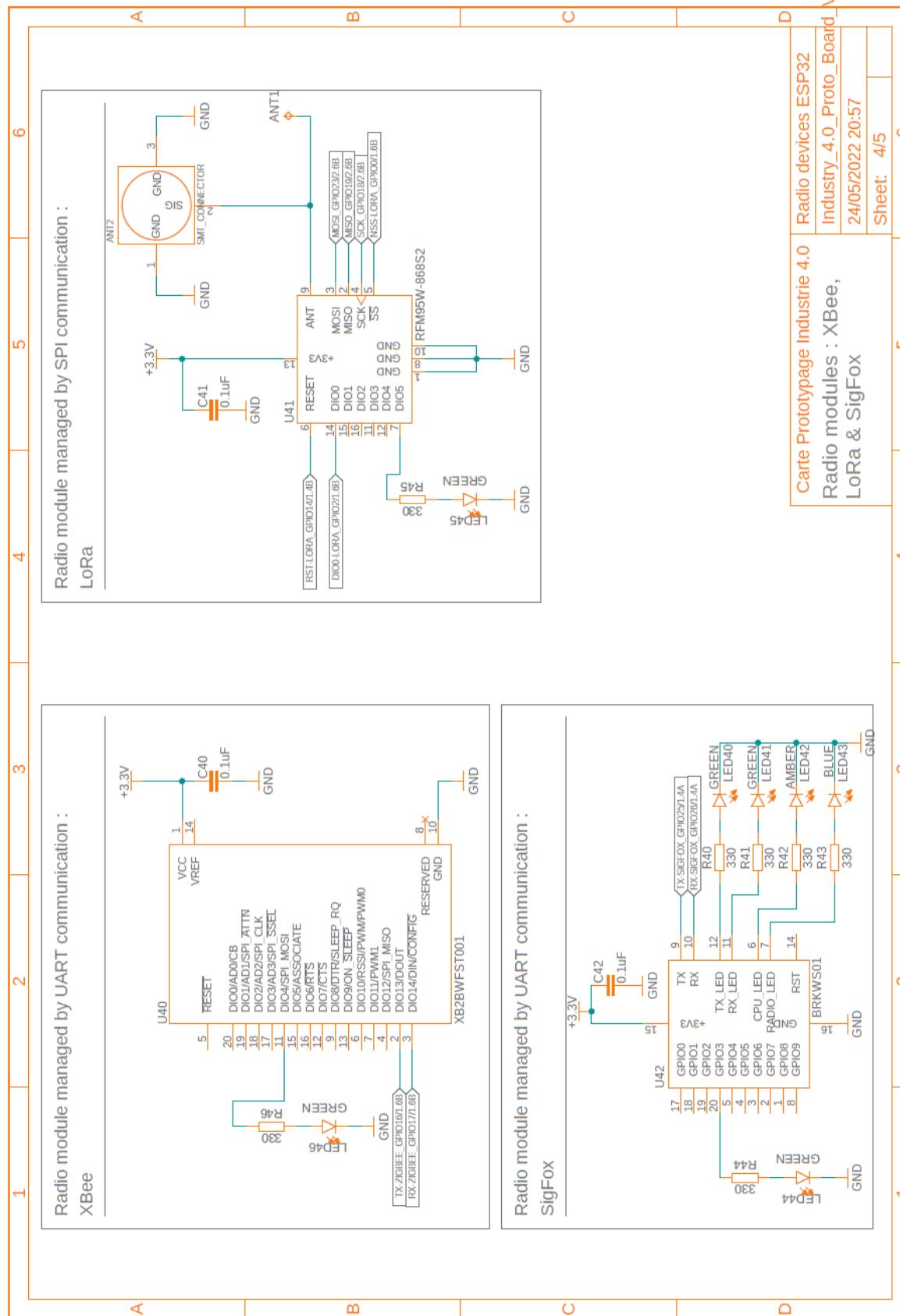
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 4 : PCB Multi-Protocole : SCHEMA page 3/5



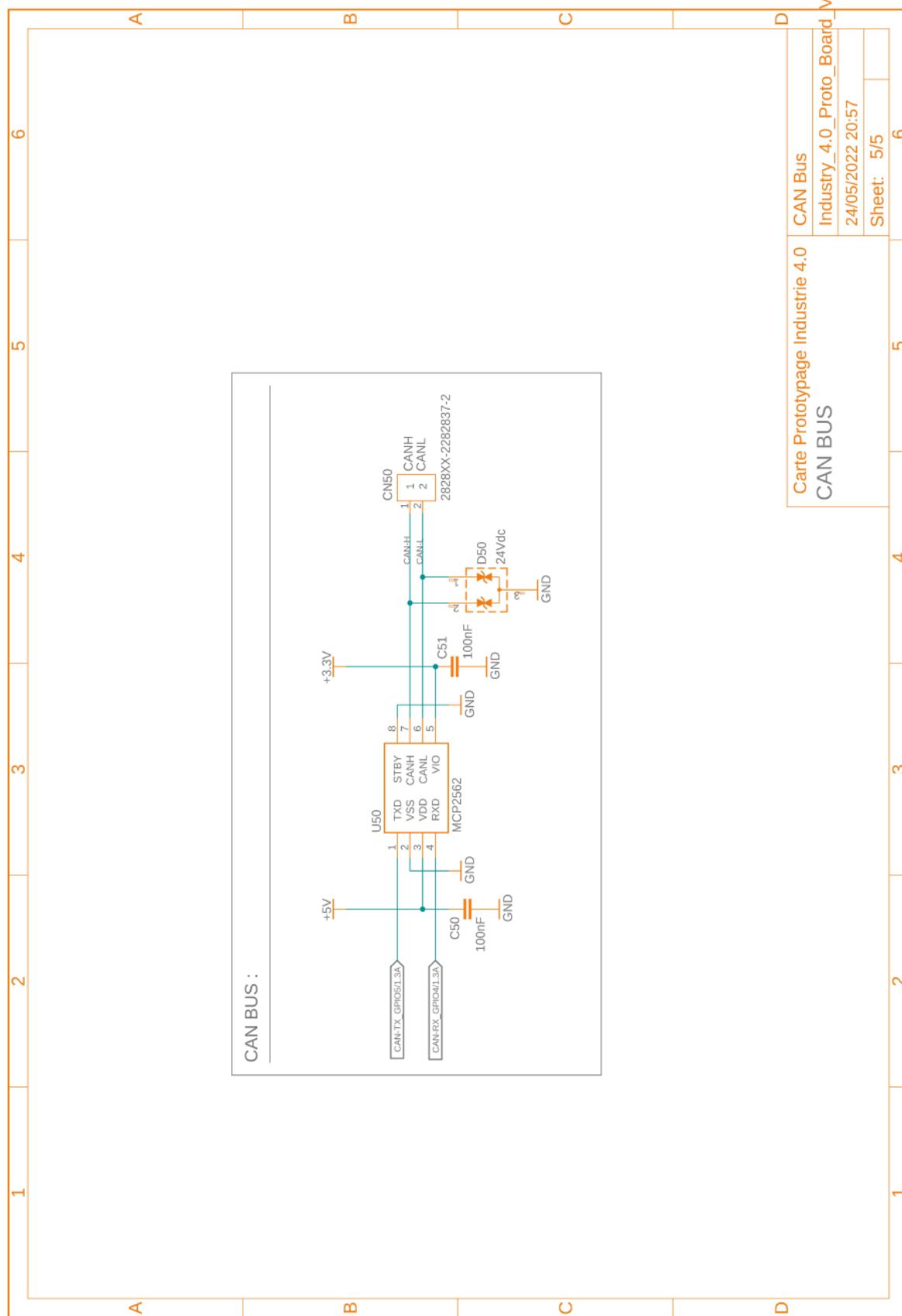
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 5 : PCB Multi-Protocole : SCHEMA page 4/5



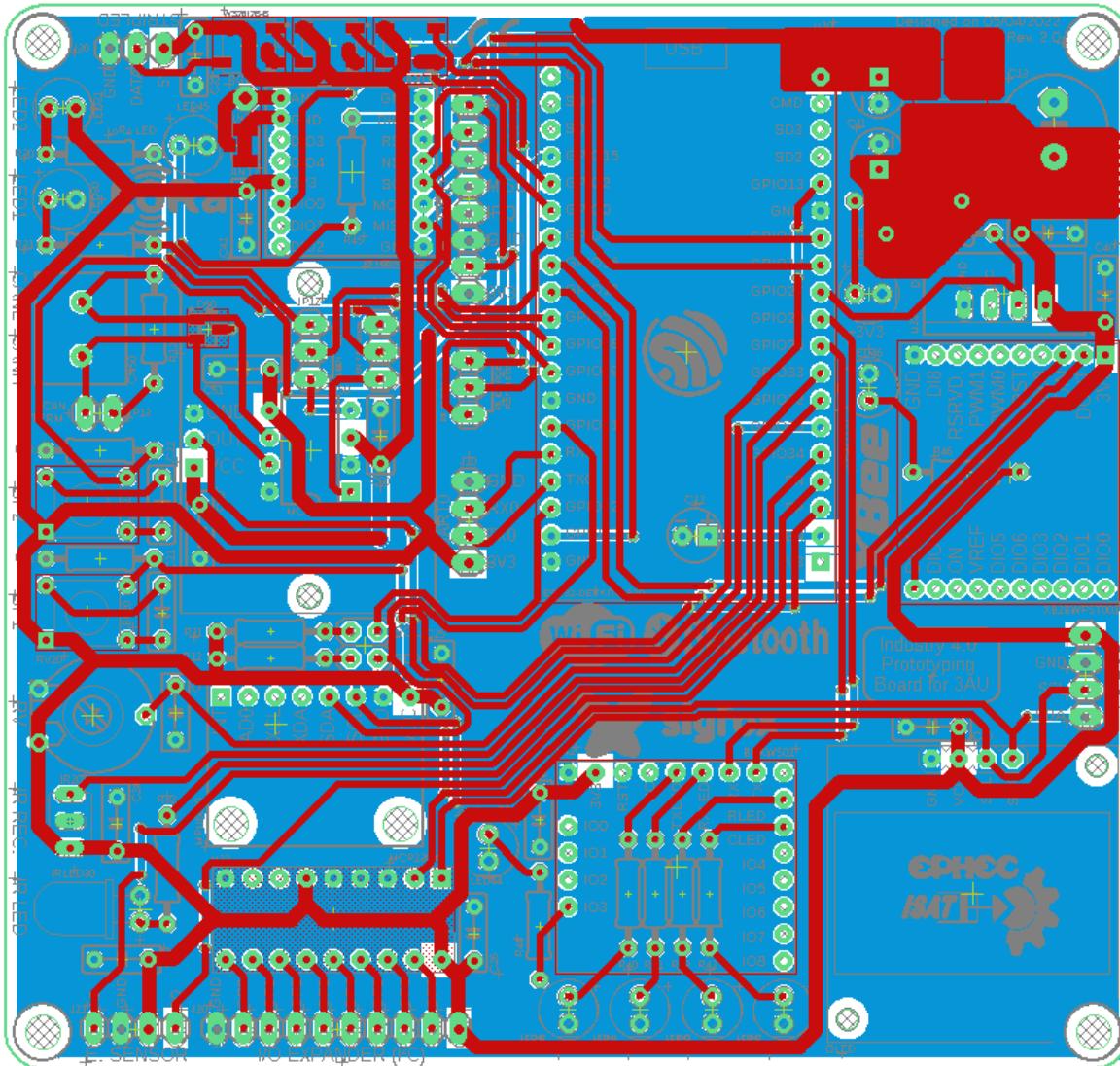
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 6 : PCB Multi-Protocole : SCHEMA page 5/5



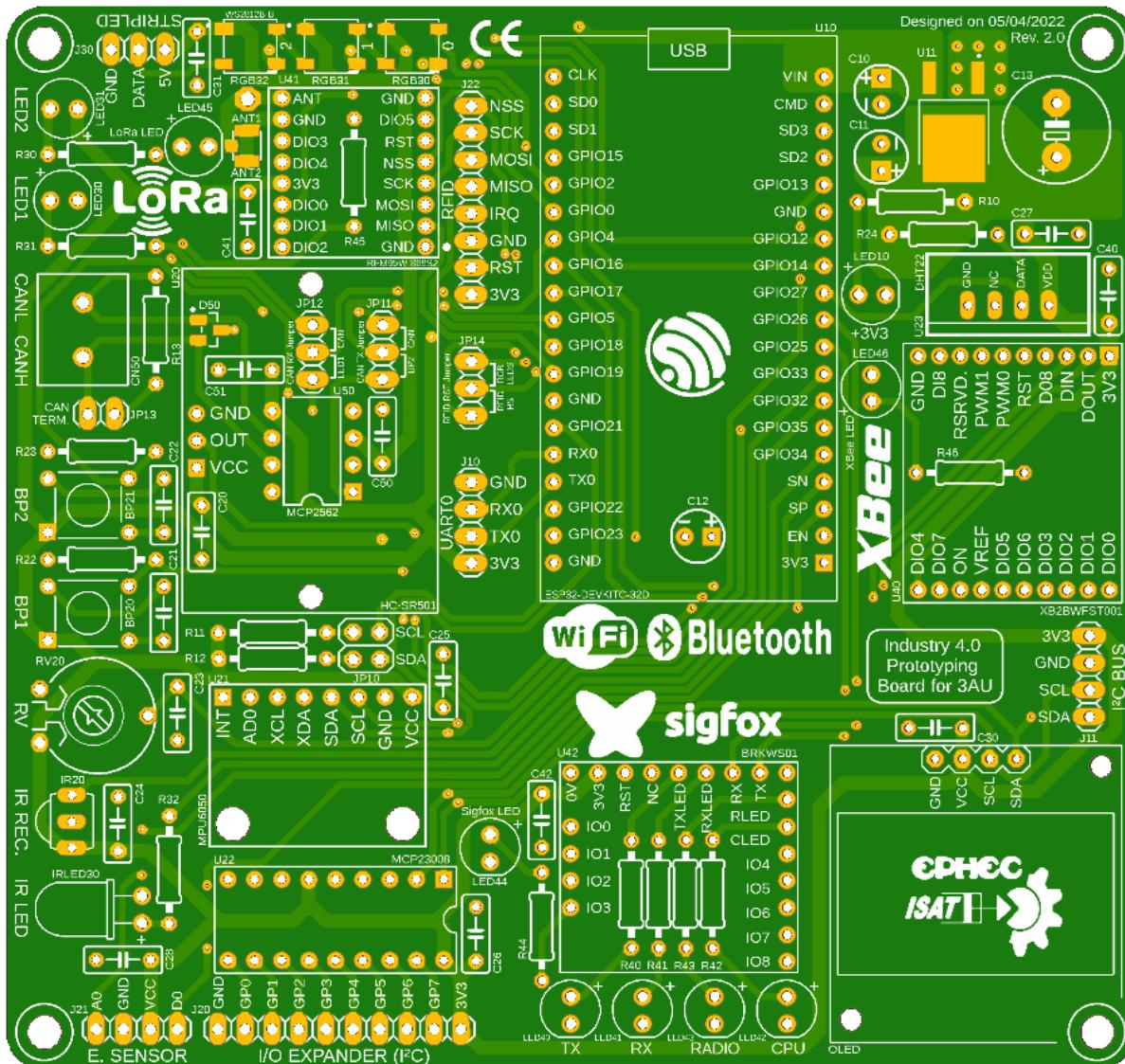
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 7 : PCB Multi-Protocole : BOARD routage



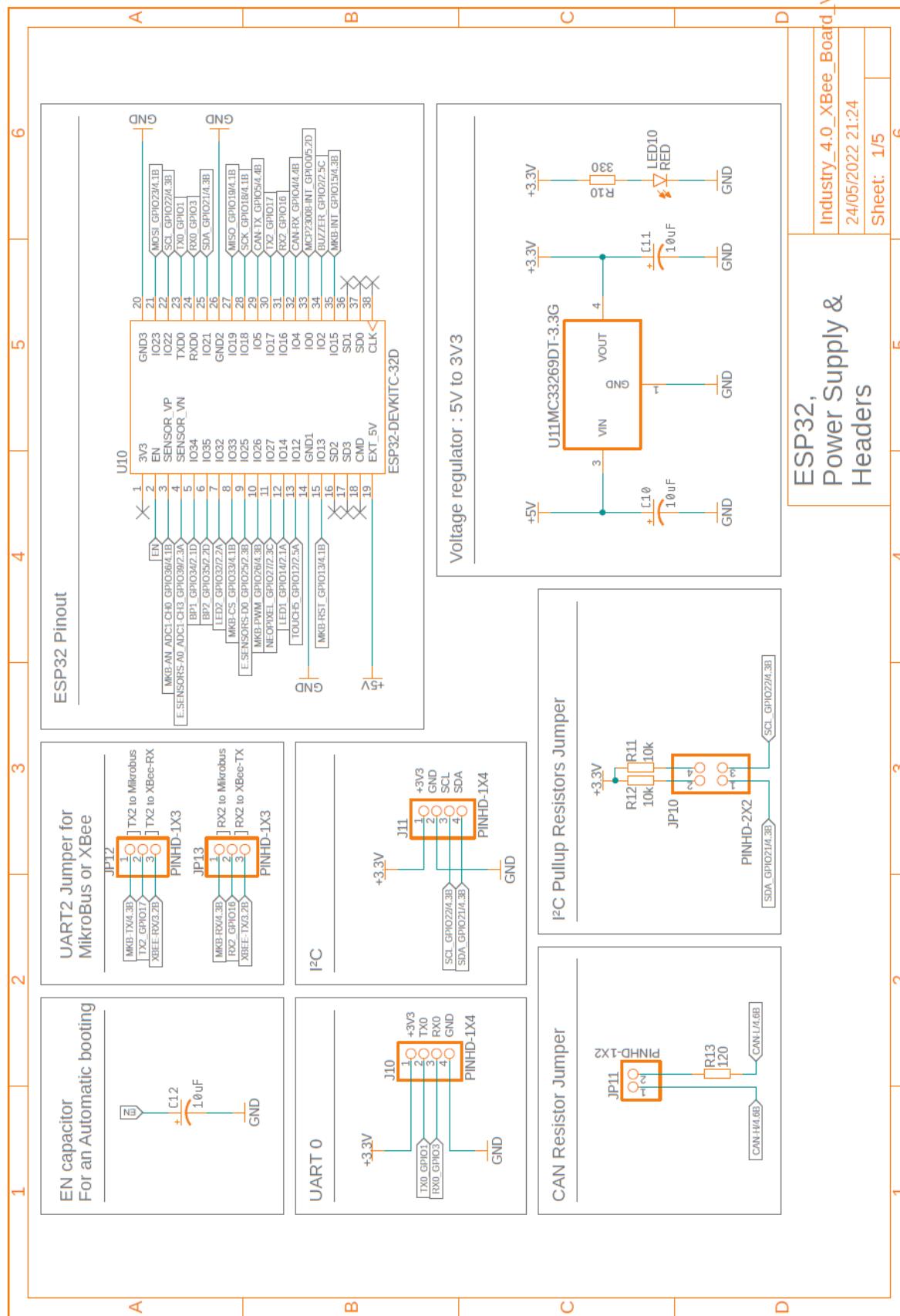
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 8 : PCB Multi-Protocole : BOARD rendu



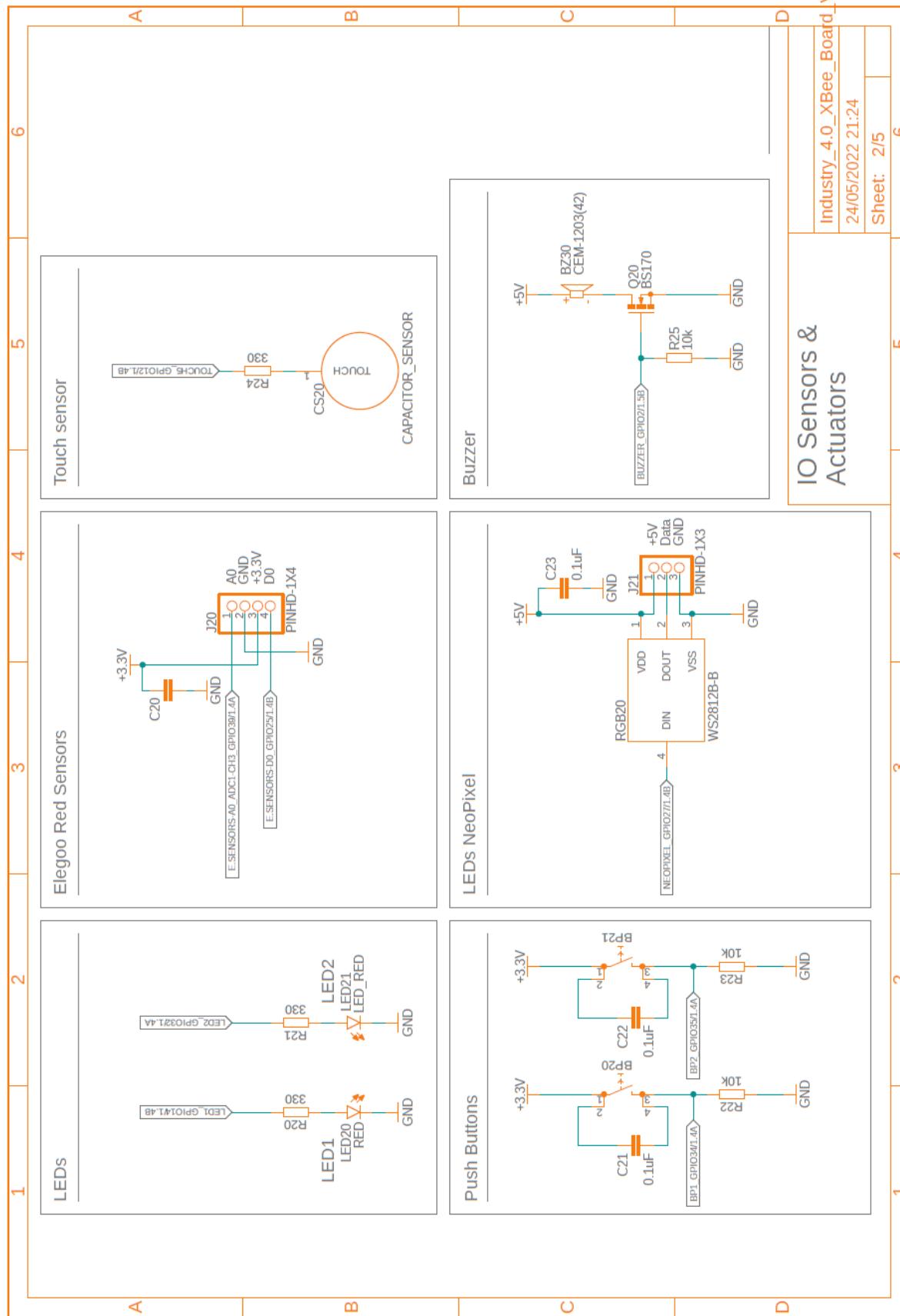
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 9 : PCB Mono-Protocole Zigbee : SCHEMA page 1/5



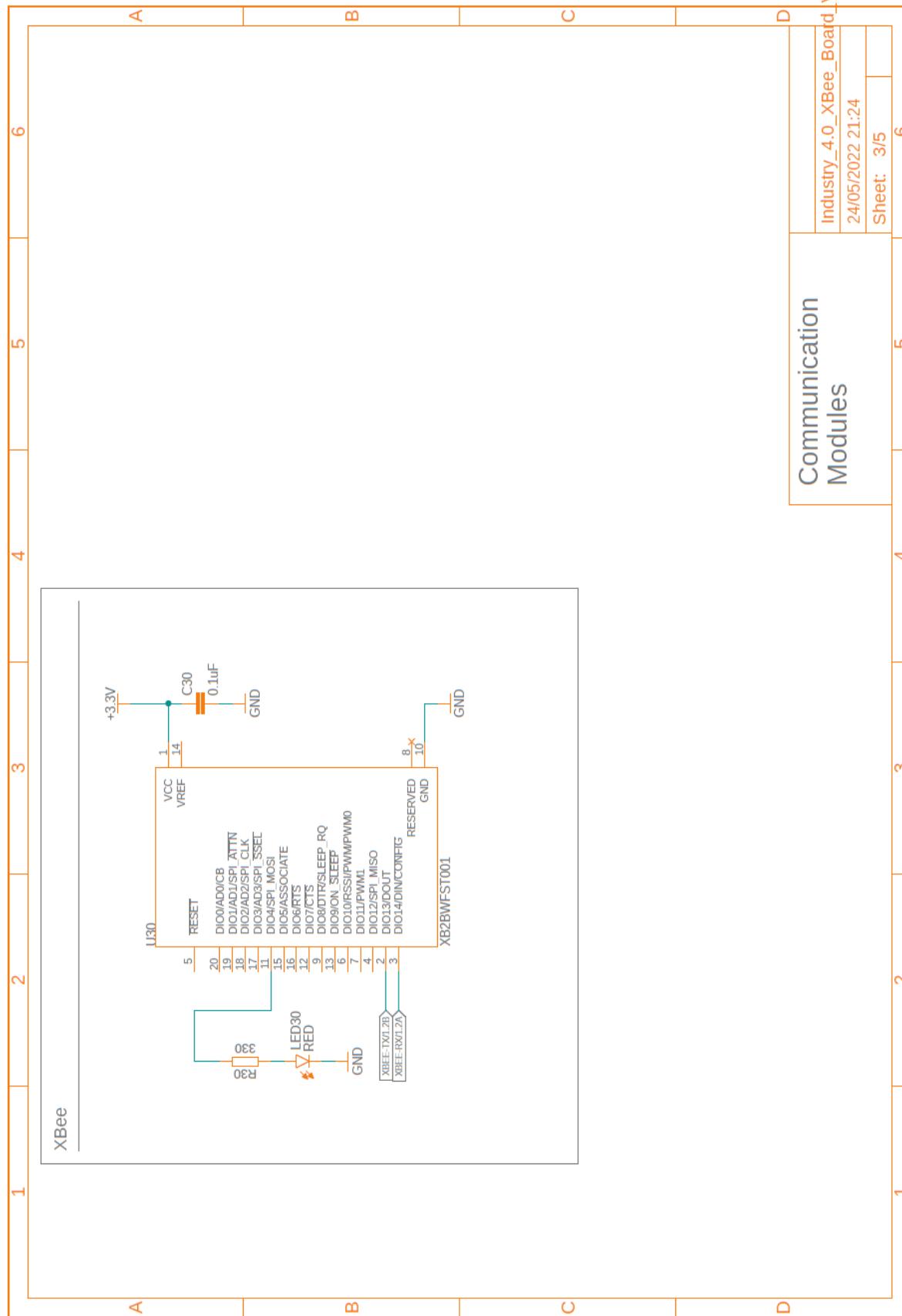
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 10 : PCB Mono-Protocole Zigbee : SCHEMA page 2/5



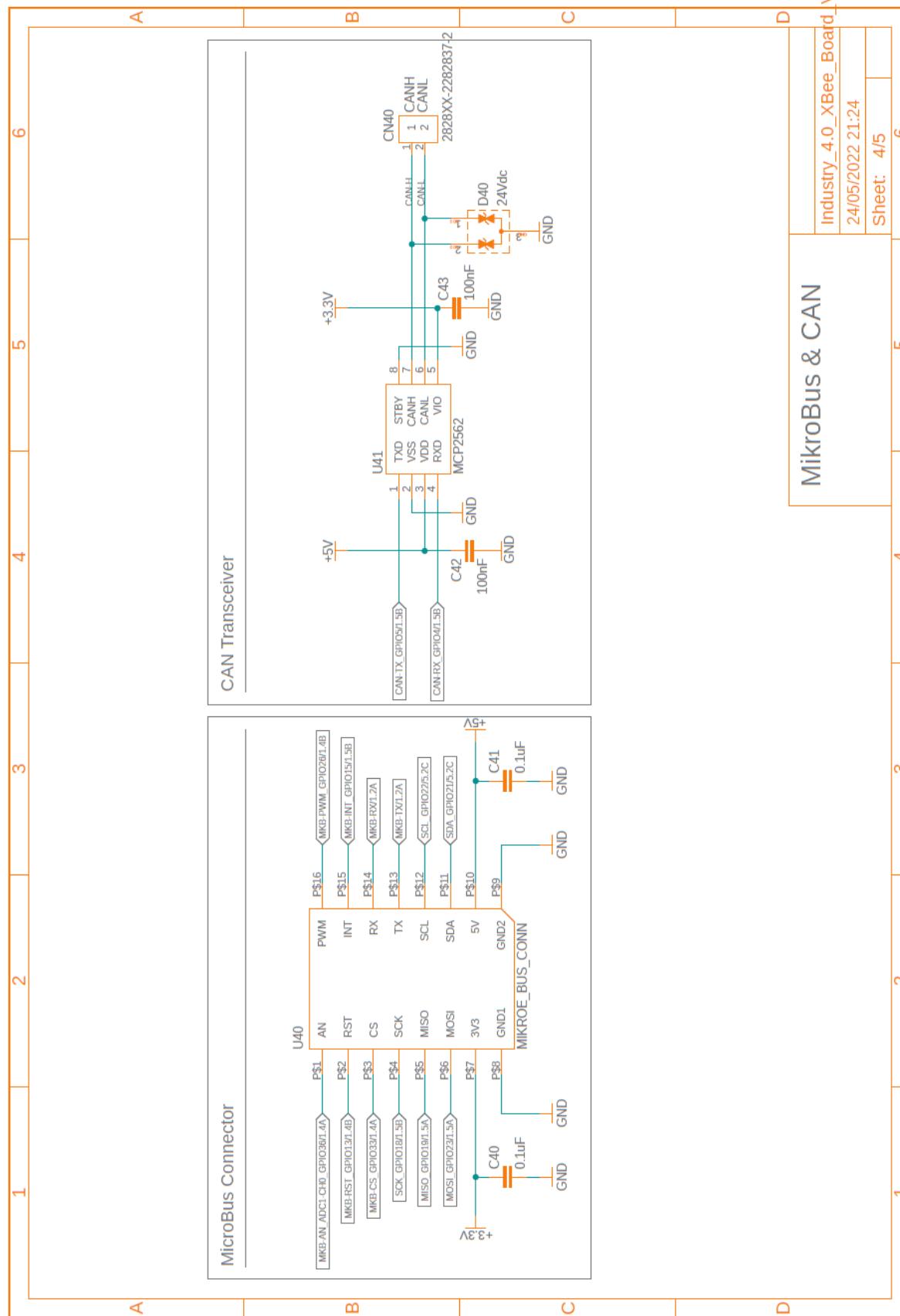
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 11 : PCB Mono-Protocole Zigbee : SCHEMA page 3/5



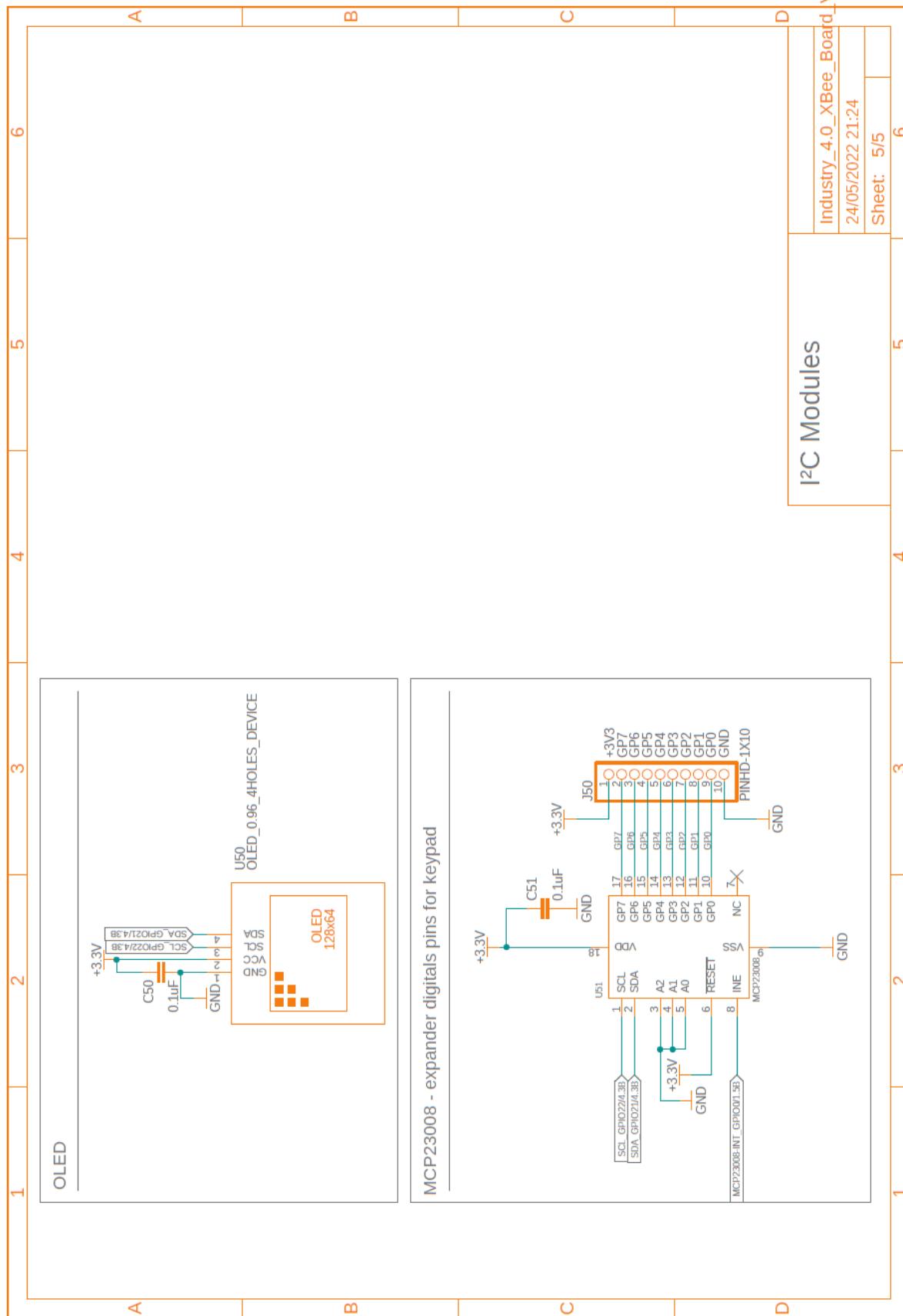
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 12 : PCB Mono-Protocole Zigbee : SCHEMA page 4/5



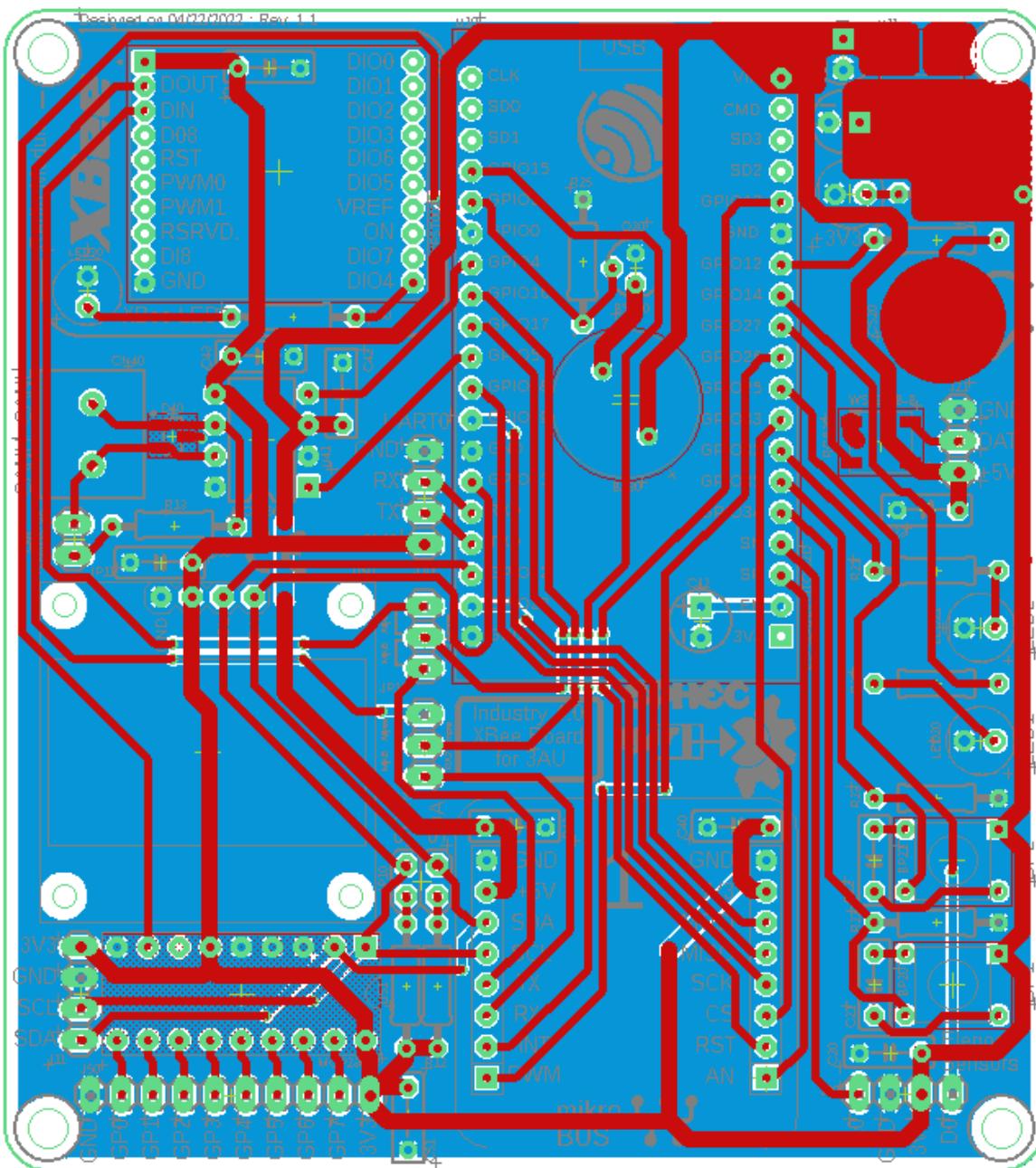
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 13 : PCB Mono-Protocole Zigbee : SCHEMA page 5/5



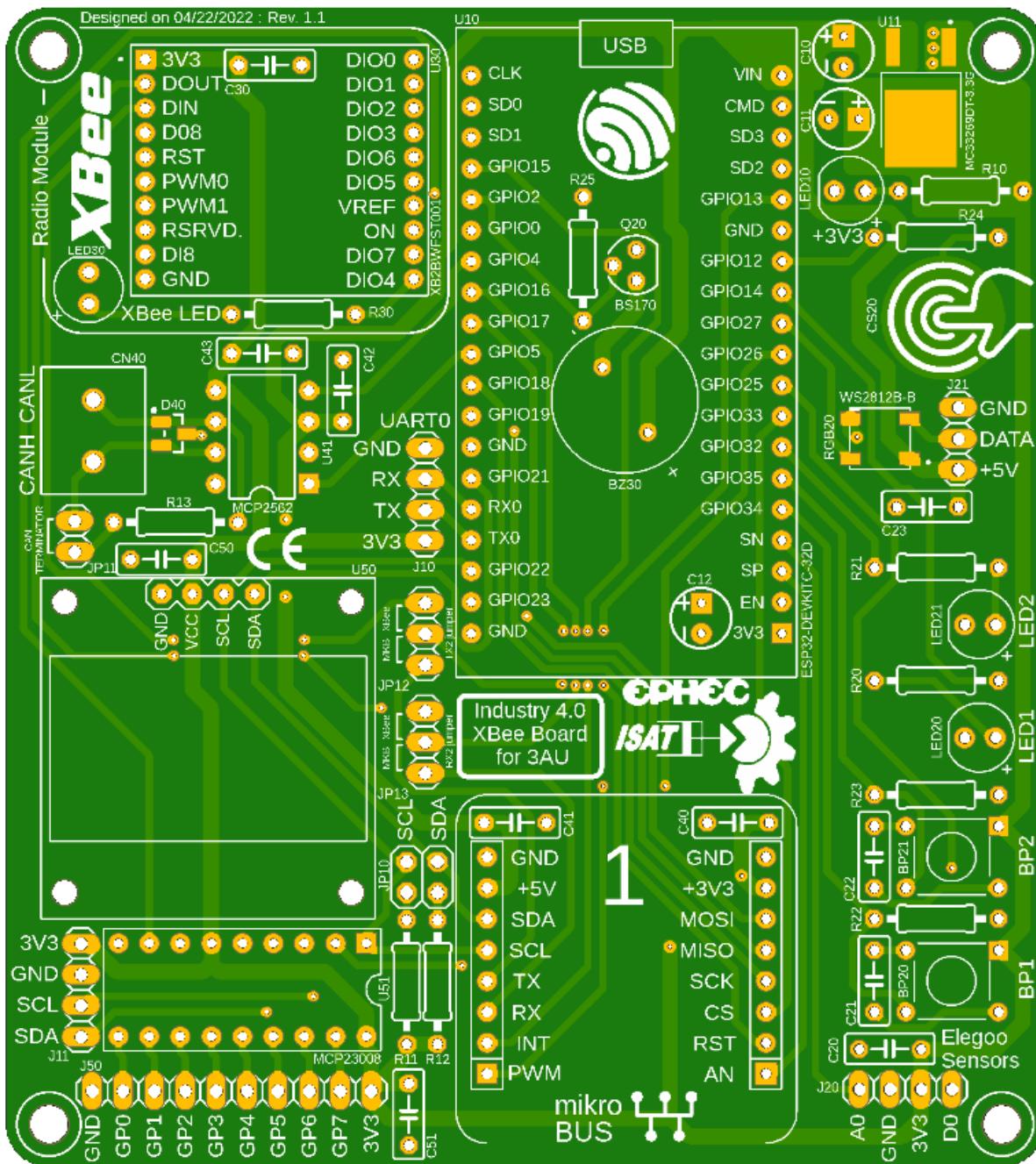
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 14 : PCB Mono-Protocole Zigbee : BOARD routage



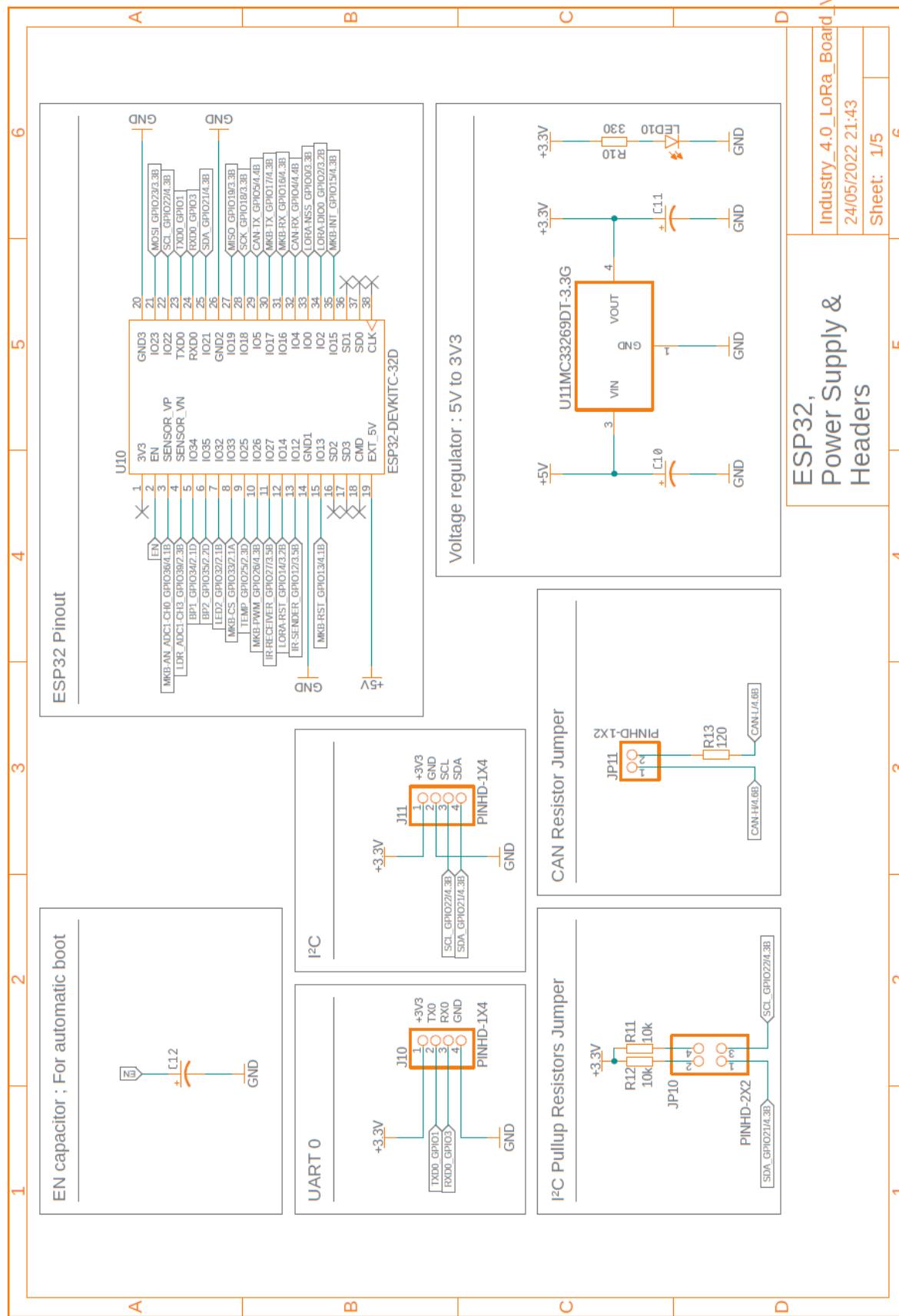
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 15 : PCB Mono-Protocole Zigbee : BOARD rendu



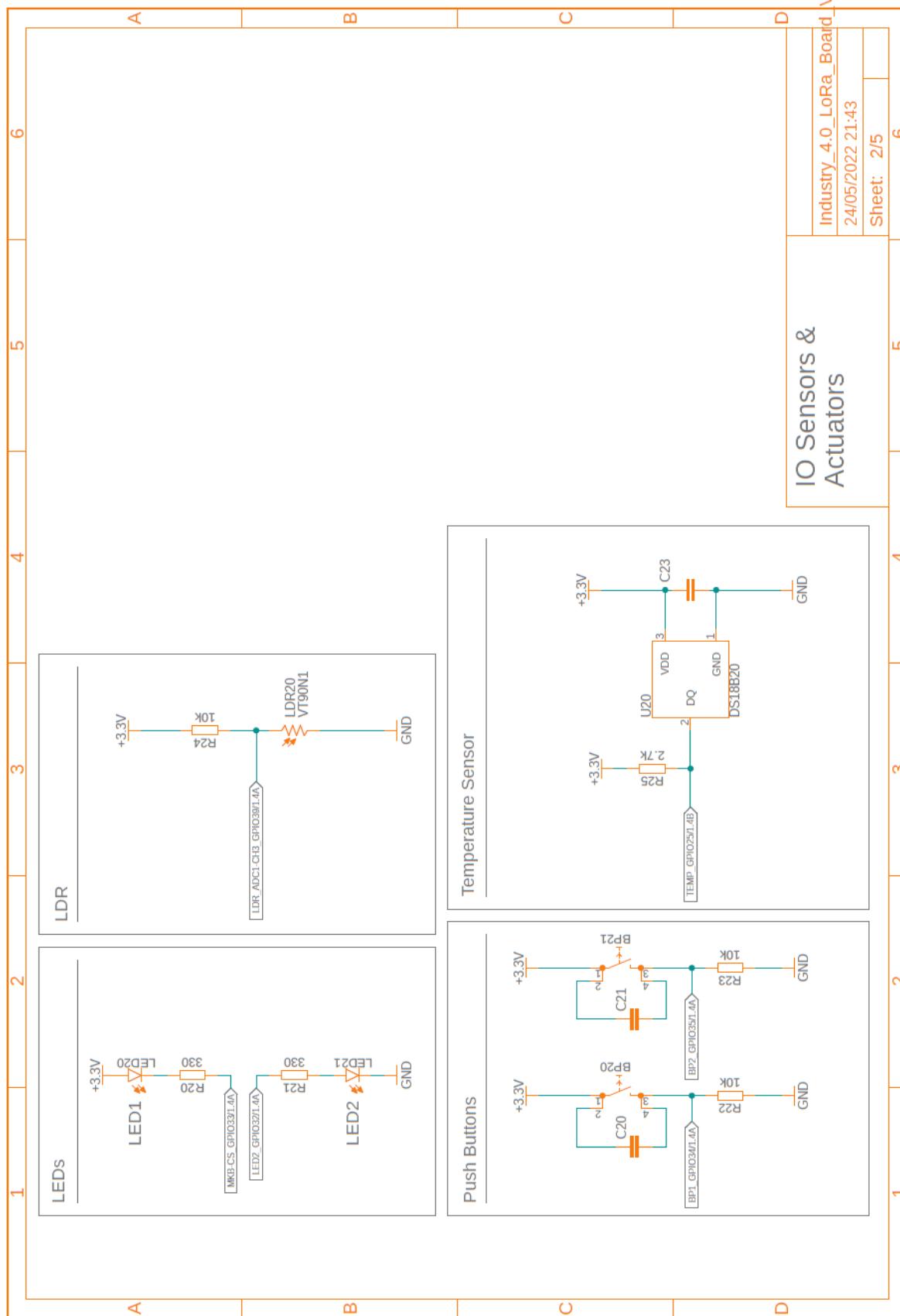
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 16 : PCB Mono-Protocole LoRa : SCHEMA page 1/5



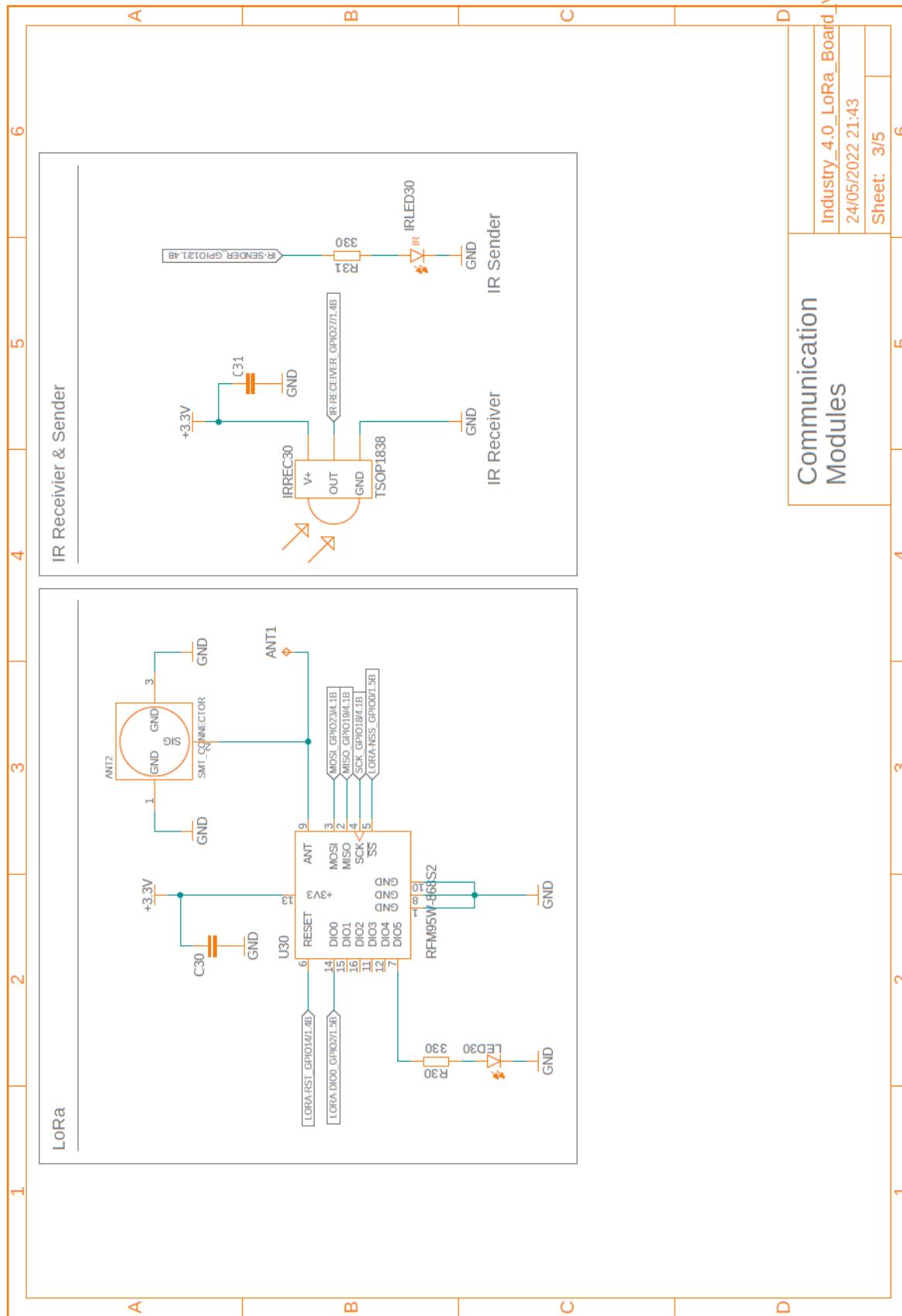
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 17 : PCB Mono-Protocole LoRa : SCHEMA page 2/5



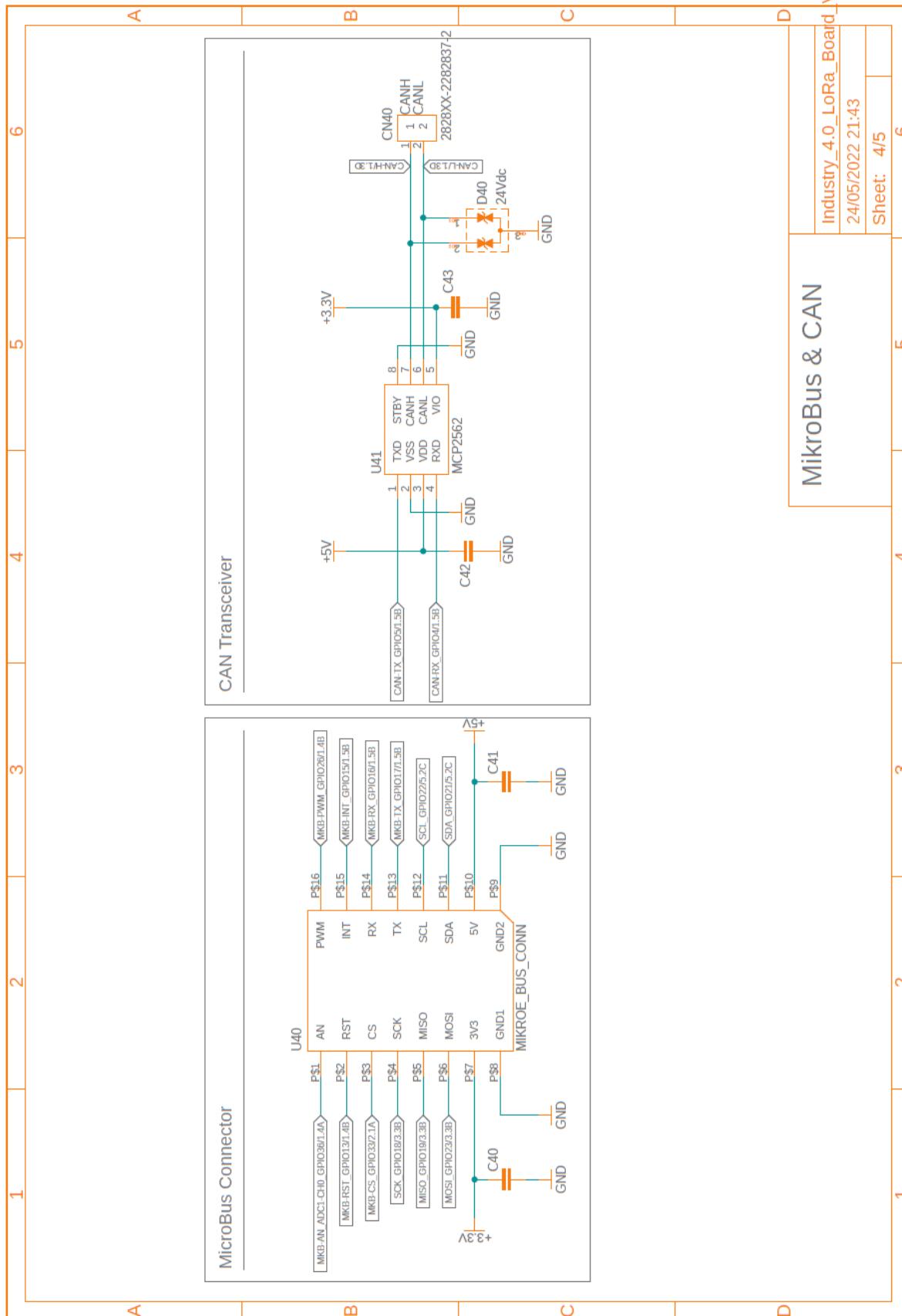
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 18 : PCB Mono-Protocole LoRa : SCHEMA page 3/5



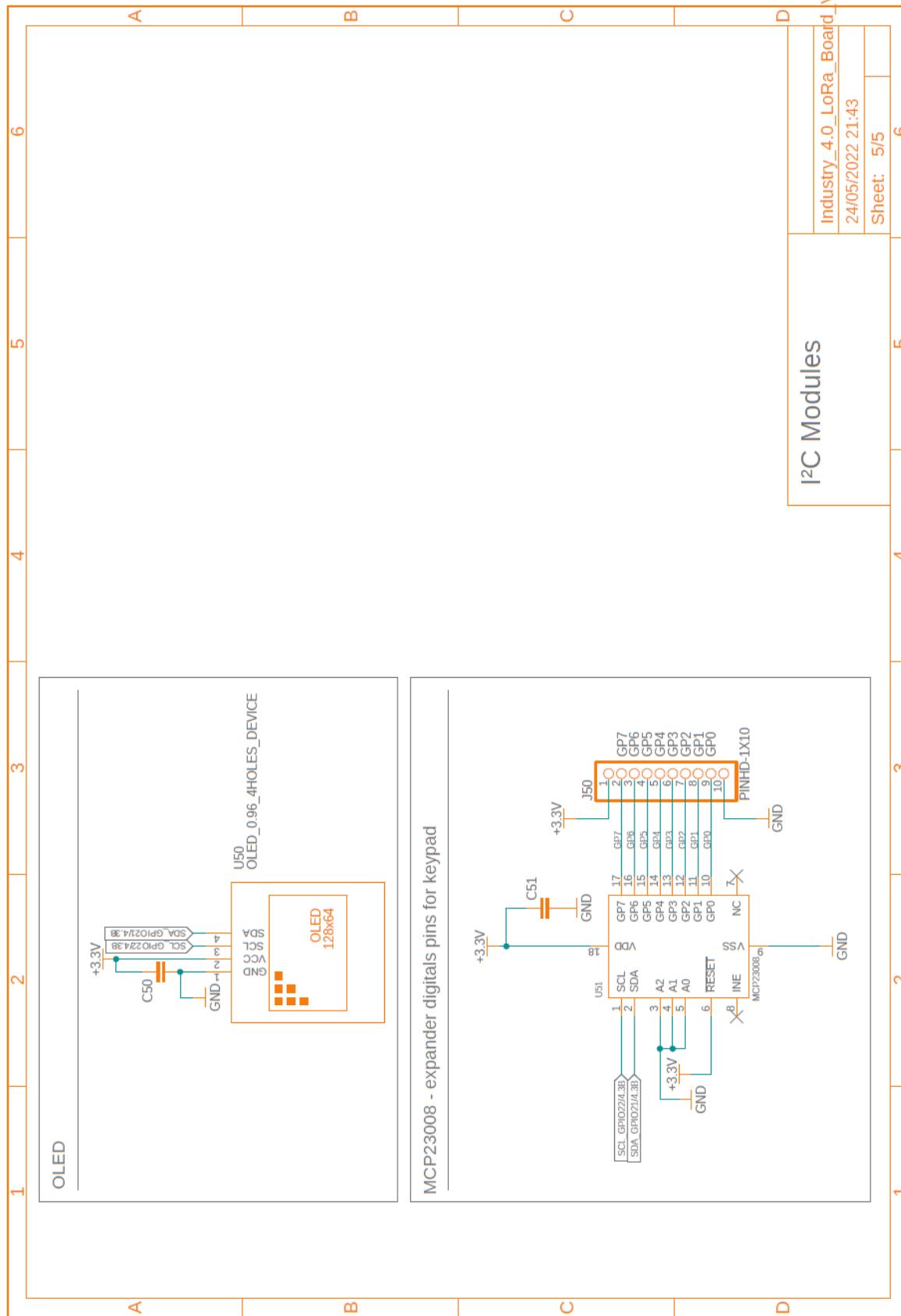
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 19 : PCB Mono-Protocole LoRa : SCHEMA page 4/5



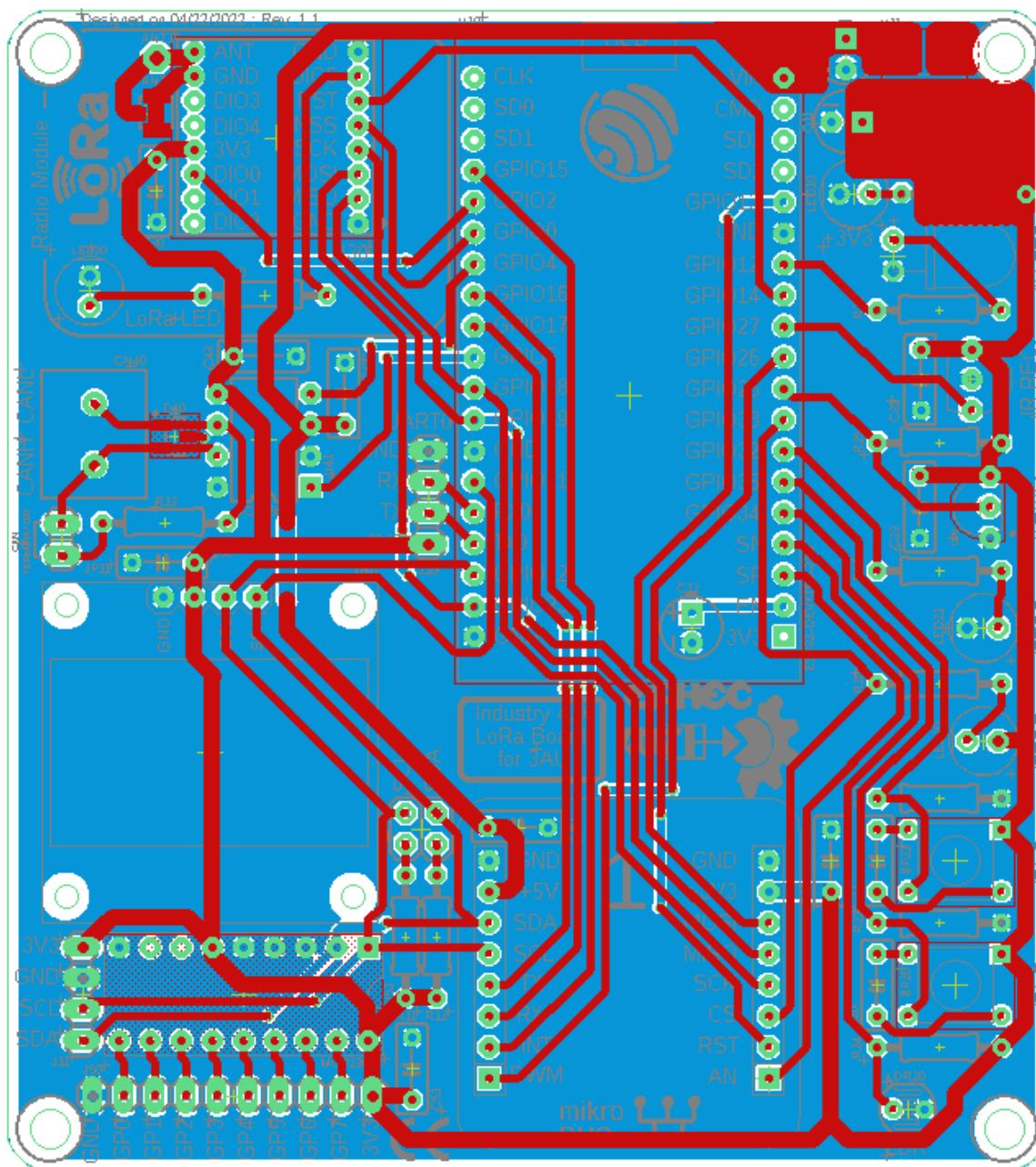
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 20 : PCB Mono-Protocole LoRa : SCHEMA page 5/5



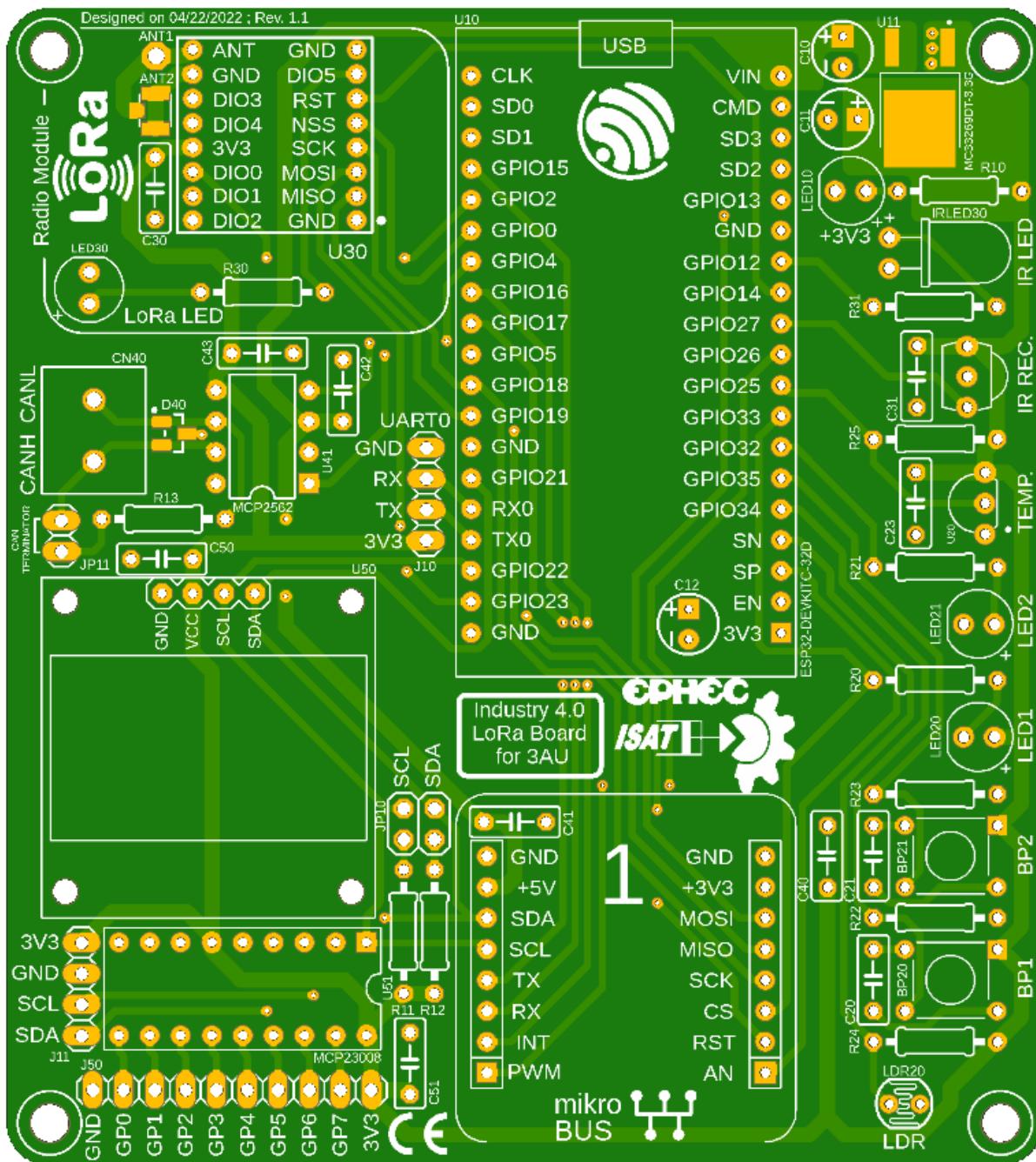
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 21 : PCB Mono-Protocole LoRa : BOARD routage



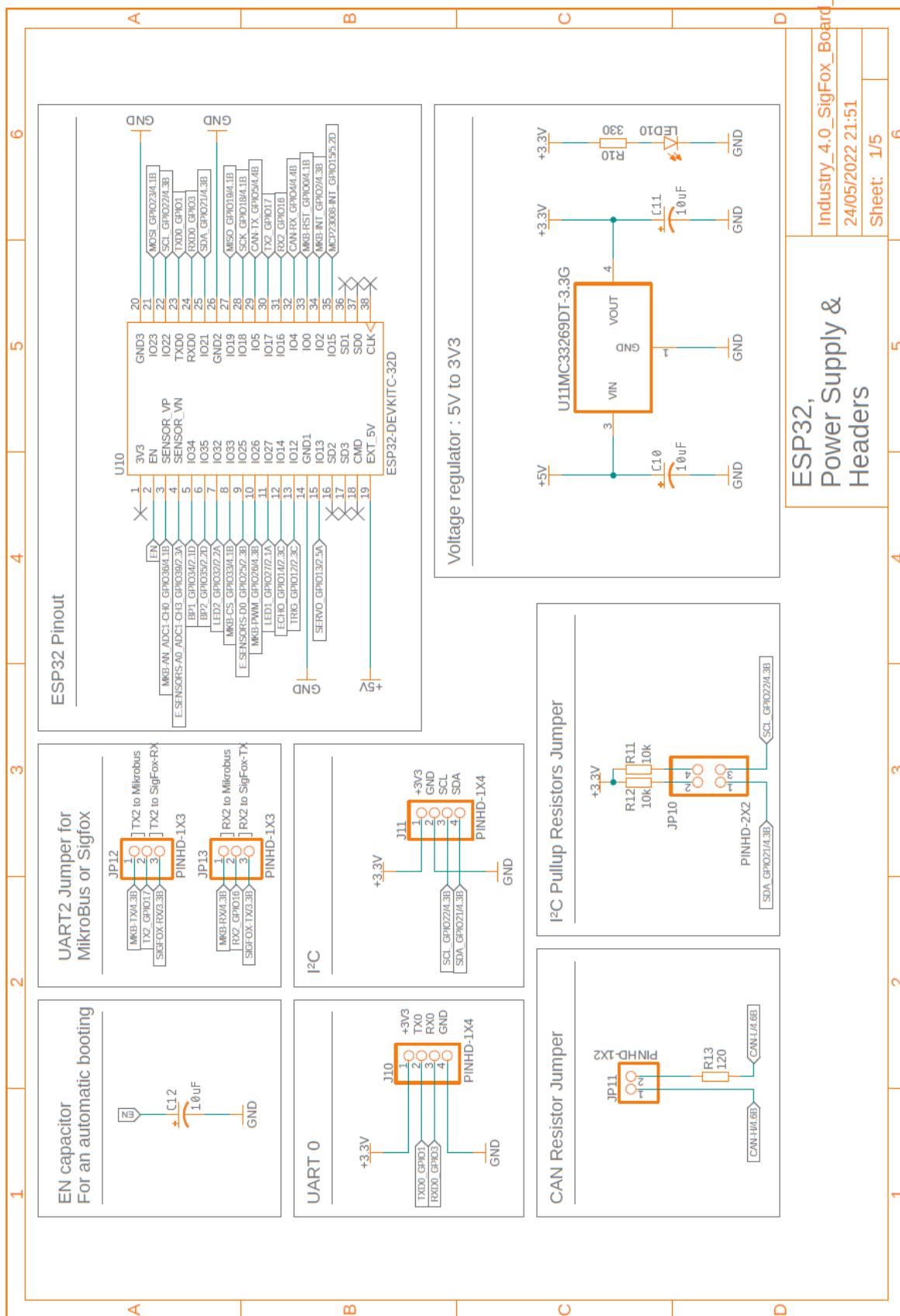
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 22 : PCB Mono-Protocole LoRa : BOARD rendu



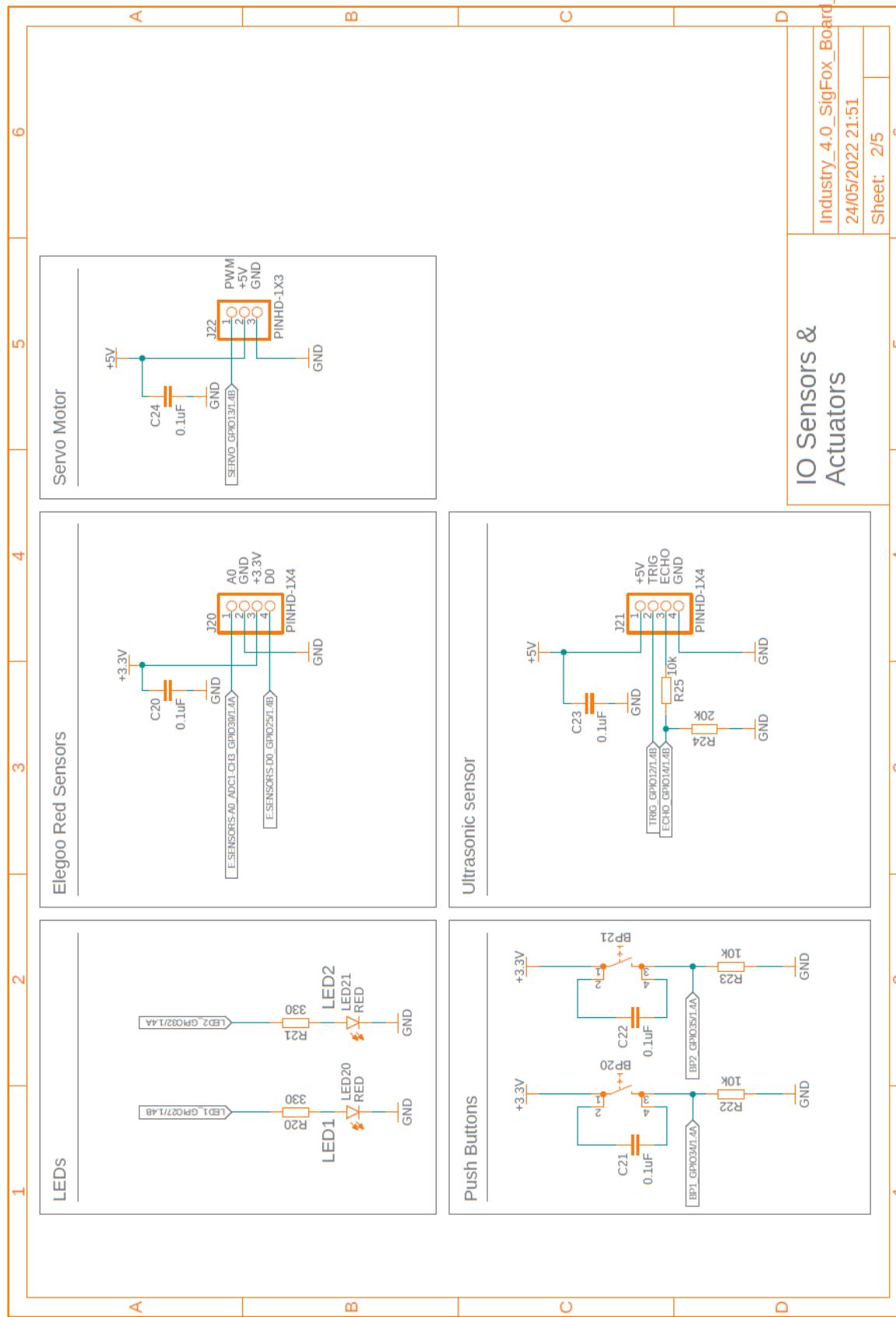
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 23 : PCB Mono-Protocole SigFox : SCHEMA page 1/5



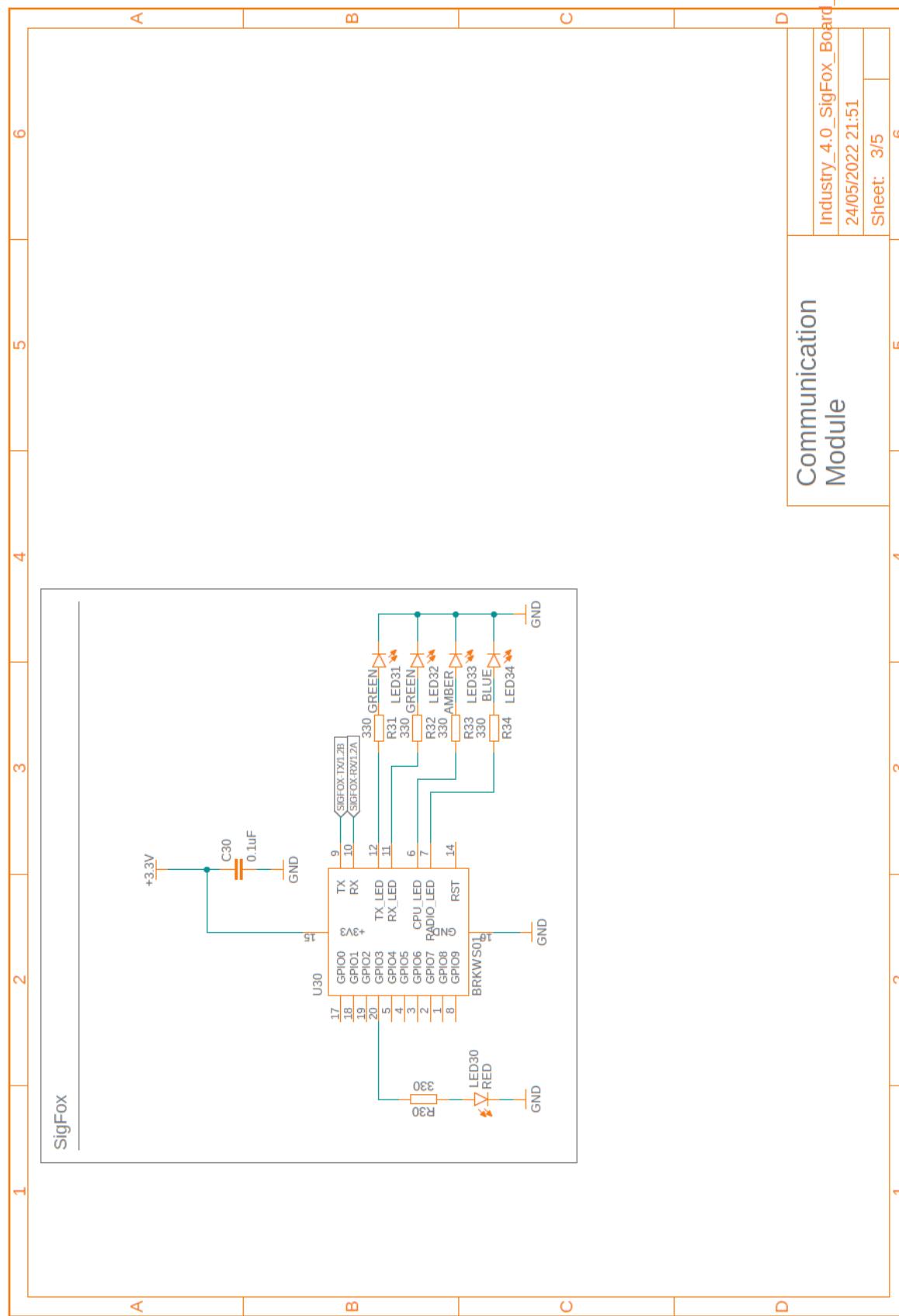
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 24 : PCB Mono-Protocole SigFox : SCHEMA page 2/5



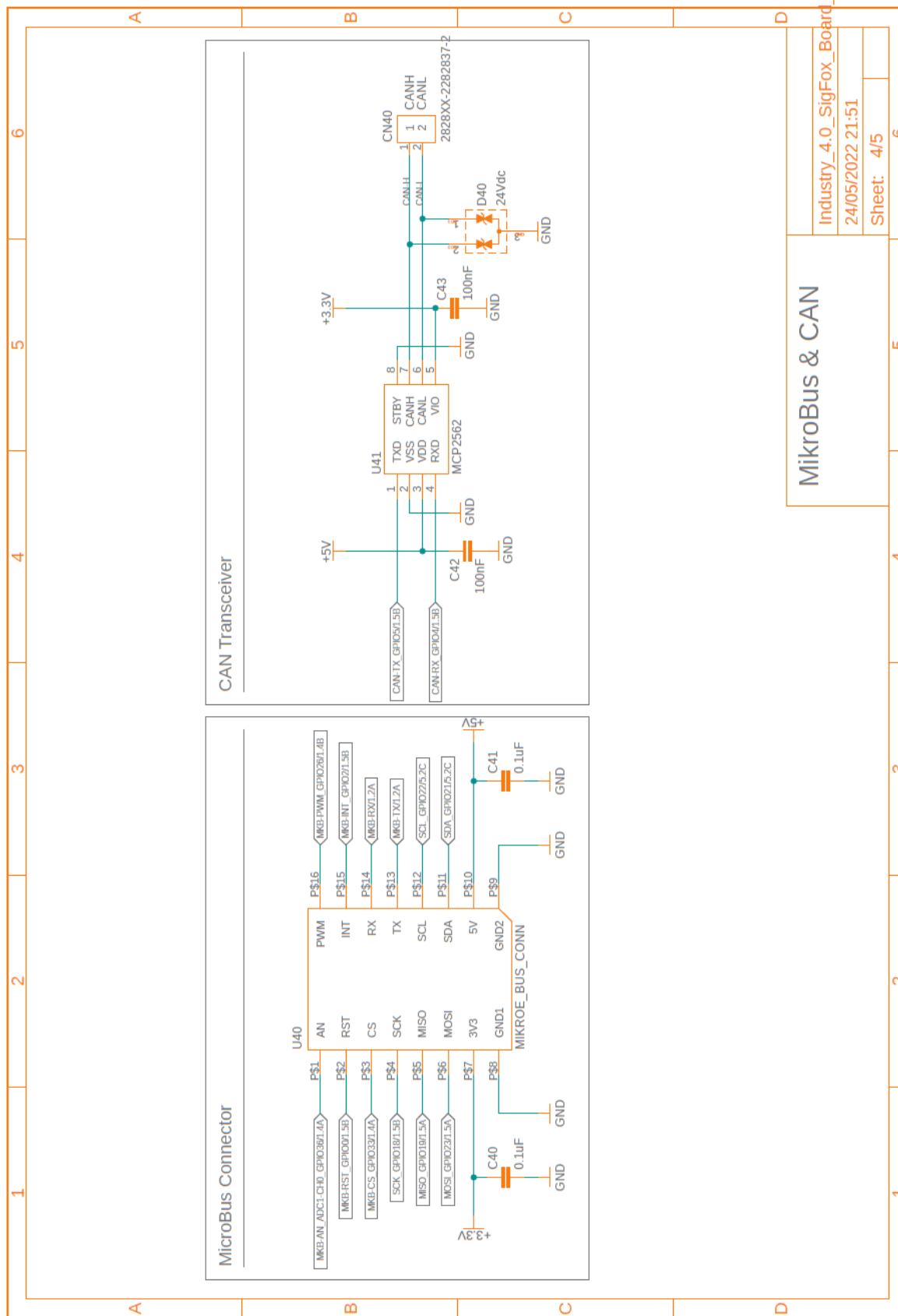
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 25 : PCB Mono-Protocole SigFox : SCHEMA page 3/5



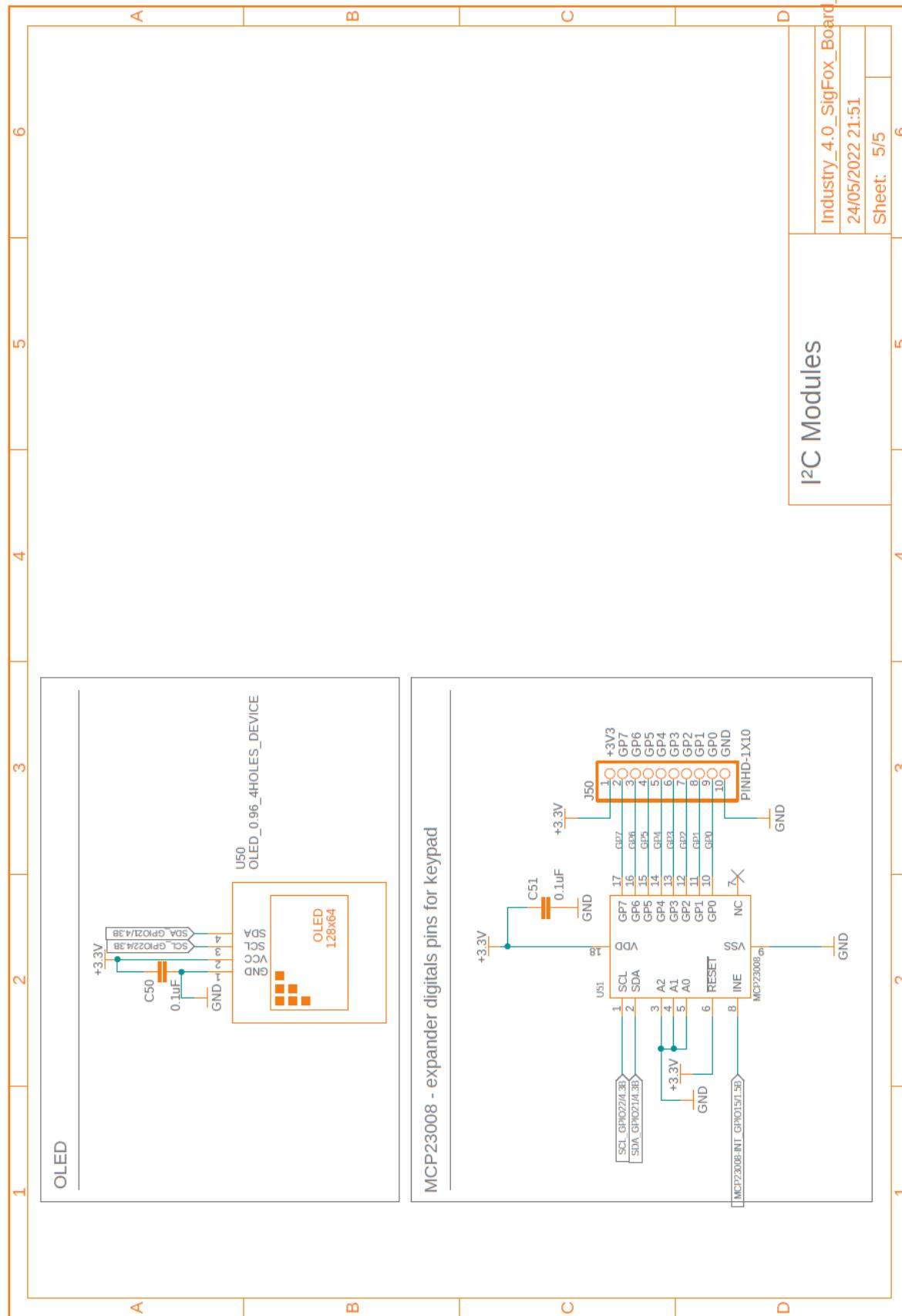
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 26 : PCB Mono-Protocole SigFox : SCHEMA page 4/5

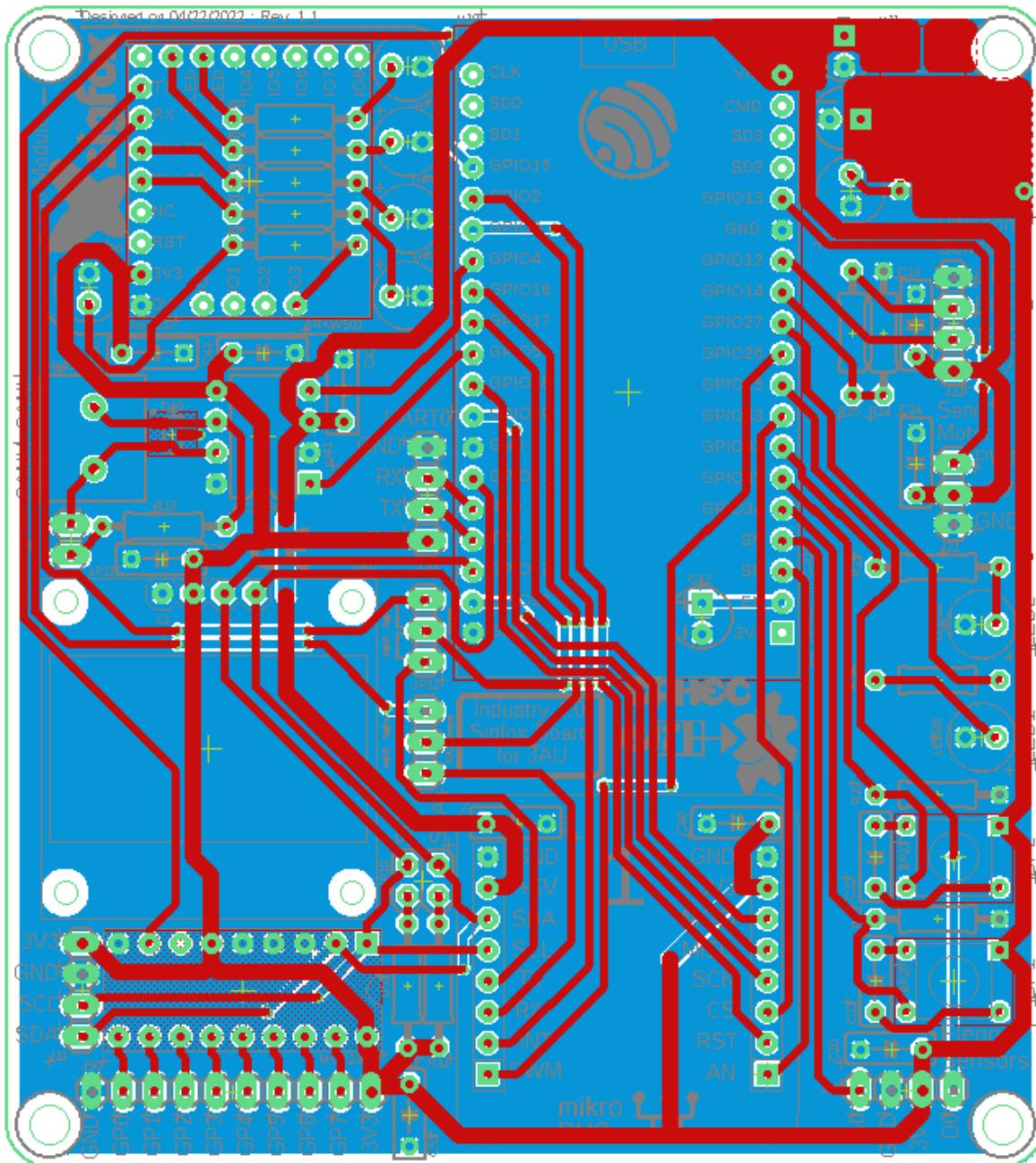


CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

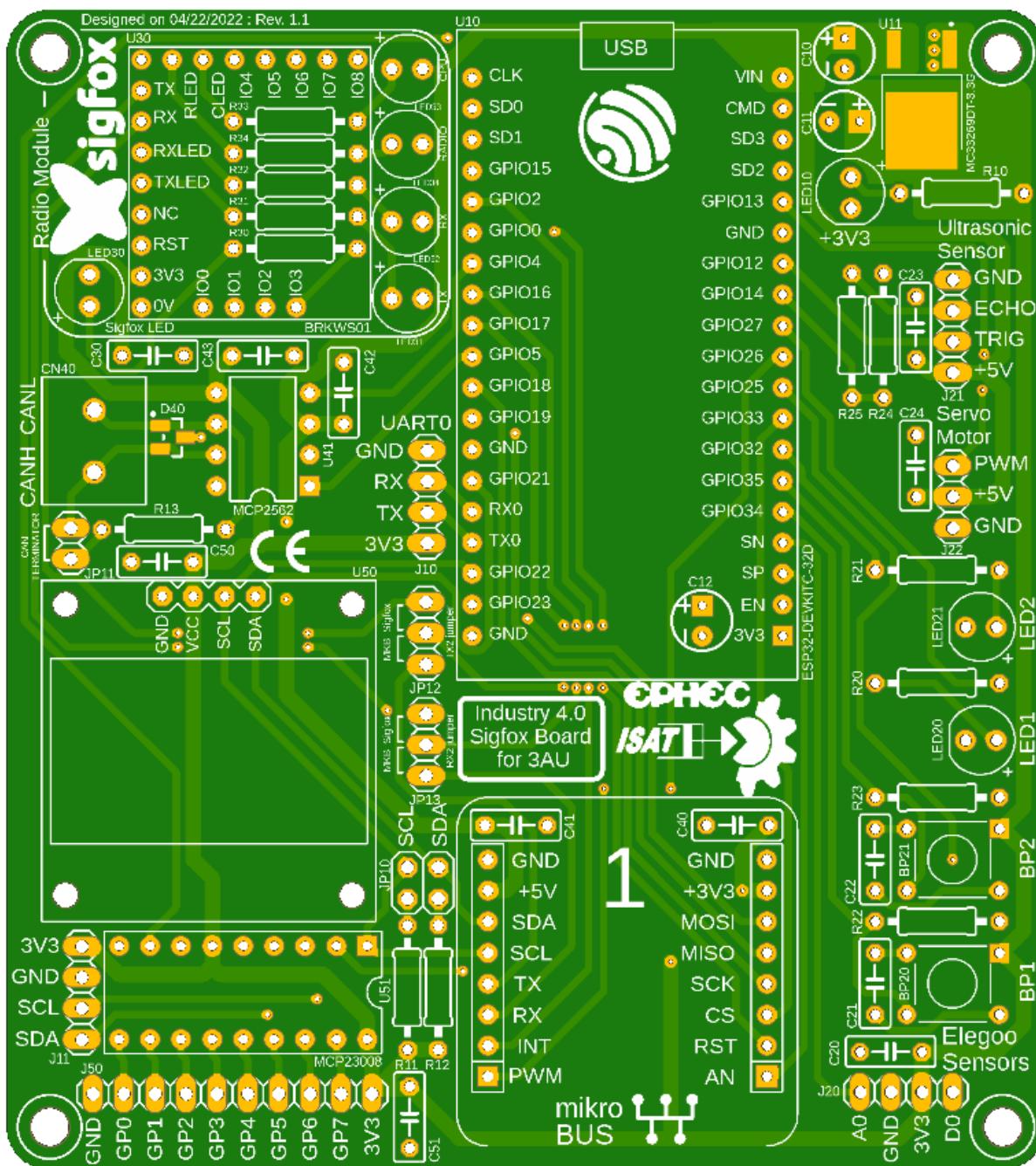
■ Annexe 27 : PCB Mono-Protocole SigFox : SCHEMA page 5/5



■ Annexe 28 : PCB Mono-Protocole SigFox : BOARD routage

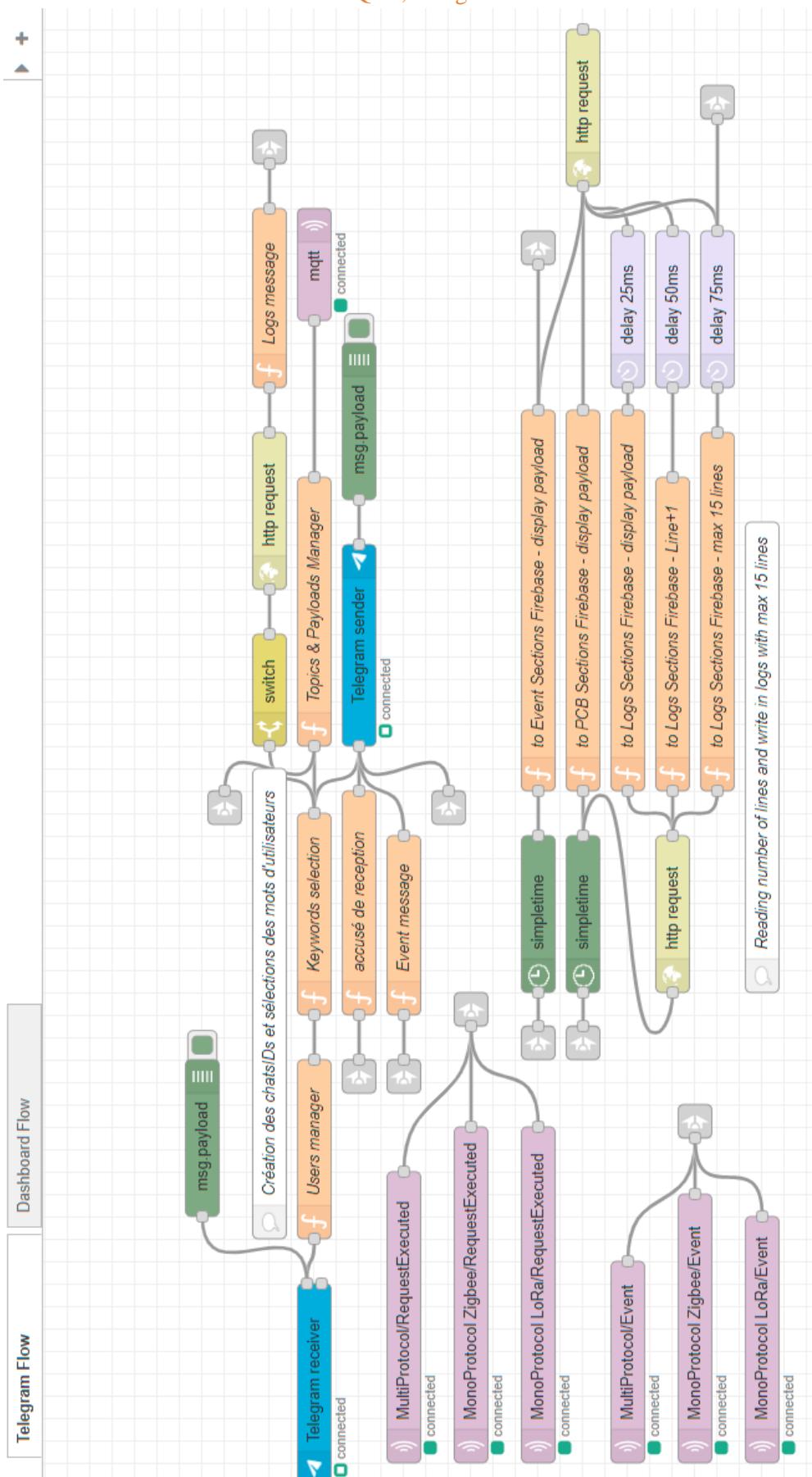


■ Annexe 29 : PCB Mono-Protocole SigFox : BOARD rendu



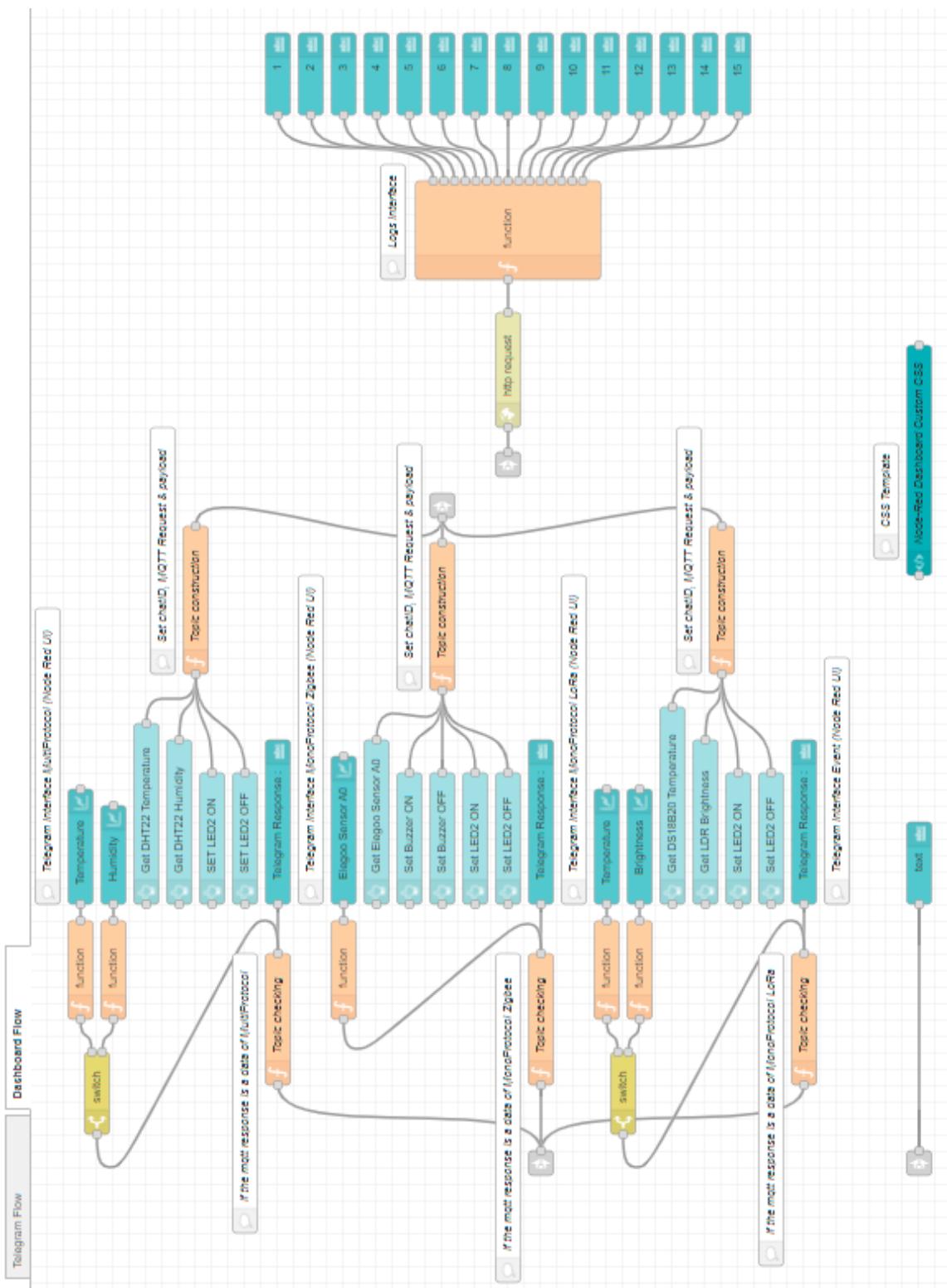
CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 30 : NodeRed : Gestion MQTT, Telegram & FireBase



CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

■ Annexe 31 : NodeRed : User Interface Dashboard



■ Annexe 32 : NodeRed : Script Users manager

```

chat_ID = msg.payload.chatId;
users = ["ISATGroup", "Davide", "Aldo", "Matteo"]
global.set("all_chatIDs", [-701587570, 5307972959, 5329300191, 5379739416]);
//Le bot répondra à celui qui lui a envoyé un msg uniquement :
for(let i = 0; i < 4; i++){
    if(chat_ID == (global.get("all_chatIDs"))[i]){
        global.set("chatID", chat_ID);
        global.set("user", users[i]);
    }
}

return msg;

```

■ Annexe 33 : NodeRed : Script Keywords selection

```

recv = msg.payload.content;

msg.payload = {}
msg.payload.options = {disable_web_page_preview : true, parse_mode : "Markdown"};
msg.payload.chatId = global.get("chatID");
msg.payload.type = "message";
// Name of each PCB           Name of each Button          Button position (line)
var myDictKeyboard = {
    "MultiProtocol" : [ "Get DHT22 Temperature", 1,
                        "Get DHT22 Humidity", 2,
                        "Set LED2 ON", 3,
                        "Set LED2 OFF", 3],
    "MonoProtocol Zigbee" : [ "Get Elegoo Sensor A0", 1,
                               "Set Buzzer ON", 2,
                               "Set Buzzer OFF", 2,
                               "Set LED2 ON", 3,
                               "Set LED2 OFF", 3],
    "MonoProtocol LoRa" : [ "Get DS18B20 Temperature", 1,
                            "Get LDR Brightness", 2,
                            "Set LED2 ON", 3,
                            "Set LED2 OFF", 3]
};

//-----MENU KEYWORD-----
if ((recv.toUpperCase() == "MENU") || (recv.toUpperCase() == "BACK TO MENU")){
    msg.payload.content = "💬 Welcome in the *Menu* "+ global.get("user")+" !
Choose an option :";

    opts = {
        reply_markup: JSON.stringify({
            keyboard: [
                ["Go to PCB MultiProtocol"],
                ["Go to PCB MonoProtocol Zigbee"],
                ["Go to PCB MonoProtocol LoRa"],
                ["Logs", "Help"]],
                'resize_keyboard' : true,
                'one_time_keyboard' : true
            ],
            disable_web_page_preview : true,

```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```

        parse_mode : "Markdown"
    };
    msg.payload.options = opts;
}

//-----STATE KEYWORD-----
else if (recv.toUpperCase() == "LOGS"){
    msg.payload.content = "📝 I will try to read the history message. Wait a
moment..."; //en cours de développement...
}
//-----HELP KEYWORD-----
else if (recv.toUpperCase() == "HELP"){
    msg.payload.content = "🆘 You asked for help. \n\r\n\r"; //en cours de
développement...
    msg.payload.content += "To enter in the main menu write and send the keyword
\"menu\". \n\r\n\r";
    msg.payload.content += "You will then be able to access all the PCBs available
in your mqtt local network. \n\r\n\r";
    msg.payload.content += "You will also be able to read the message history with
logs of a data base. \n\r\n\r";
}
//-----UNKNOWN KEYWORD-----
else {
    randomNumber = Math.round(Math.random()*5);
    msg.payload.content = ("⚠️ ");
    //msg.payload.content += randomNumber.toString();
    switch(randomNumber) {
        case 5 : msg.payload.content += " Your keyword is not in my data base";
break;
        case 4 : msg.payload.content += " I do not understand"; break;
        case 3 : msg.payload.content += " Try again with another keyword please";
break;
        case 2 : msg.payload.content += " Hein ? Can you repeat ?"; break;
        case 1 : msg.payload.content += " Sorry ? Try the keyword *HELP*"; break;
        case 0 : msg.payload.content = "👉"; break;
    }
}

//Gestion des touches cliquées dans les menus des PCB par l'utilisateur et création
des messages de retour ainsi que des topics MQTT
for (var myPCB in myDictKeyboard){
// On parcourt tous les PCB
    for (let myMsg = 0; myMsg < myDictKeyboard[myPCB].length ; myMsg+=2){
// On parcourt tous les textes des boutons (indices pairs)

        if (recv.toUpperCase() == ("GO TO PCB " + myPCB.toUpperCase() )){
// Si j'ai cliqué sur un bouton PCB dans le menu (ex: Go To PCB MultiProtocol)
            msg.payload.content = "💬 I am in the *PCB "+ myPCB +"* ! Choose an
option :"; // Le bot me répondra avec cette phrase
            global.set("PCBselected", myPCB.toUpperCase());
// On conserve quel bouton PCB on avait cliqué (Mono/MultiProtocol...)
            keyboardList = [];
// Création d'une liste contenant les boutons accessible dans le menu du PCB
            pastpositionOfButtonInOneLine = 0;
// Préparation de la variable détectant si sur une ligne il y aura 1 ou plusieurs
boutons
        }
    }
}

```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```

columnOfLine = 0;
// Préparation de la variable positionnant le texte du deuxième bouton ou plus sur
une même ligne
for (let indiceOfButton = 0; indiceOfButton<
(myDictKeyboard[myPCB].length)/2 ; indiceOfButton++) {           // On parcourt les
indices/positions des boutons du dictionnaire

    positionOfButtonInOneLine =
myDictKeyboard[myPCB] [(indiceOfButton*2)+1];                      // On
récupère les positions de chaque boutons (ex: 1, 2, 3, 3) dans chaque ligne

    if ( positionOfButtonInOneLine > pastpositionOfButtonInOneLine) {
// Si sur la ligne je ne veux mettre qu'un bouton
        keyboardList[indiceOfButton] =
[myDictKeyboard[myPCB] [indiceOfButton*2]];                         // On récupère les
valeurs des clés (texte des btn) pour les formater sous forme de liste keyboard,
ex : [[a],[b],[c],...]
        columnOfLine = 0;
// On réinitialise la variable parce qu'il n'y a pour l'instant pas un deuxième
bouton sur la même ligne
    }

    else if (positionOfButtonInOneLine ==
pastpositionOfButtonInOneLine){                                         // Si sur la ligne je
veux mettre 2 ou plusieurs boutons
        columnOfLine++;
// Sur la même ligne il y a un deuxième bouton (ou plus), donc on incrémente la
variable
        keyboardList[indiceOfButton-columnOfLine] [columnOfLine] =
myDictKeyboard[myPCB] [(indiceOfButton*2)]; // On récupère les valeurs des clés
(texte des btn) pour les formater sous forme de liste keyboard, ex : [[a, b],[c, d,
e],[f, g],...]
    }
    pastpositionOfButtonInOneLine = positionOfButtonInOneLine;
// On récupère la précédente valeur de la position du bouton (1,2,3,3)
}
keyboardList = keyboardList.filter(n => n);
// Il y a un emplacement vide qui apparaît dans la liste parfois, donc on le filtre
(un défaut de pointeur)
keyboardList[keyboardList.length]= ["Back to Menu"];
// On rajout un dernier bouton tout en bas, celui de revenir vers le menu

opts = {                                                 // Début de création des options du
custom keyboard
    reply_markup: JSON.stringify({          // Préparation du JSON
        keyboard: keyboardList,             // On reprend la liste formatée
pour l'insérer dans le JSON (elle va permettre de positionner les boutons comme
précédemment formaté sous la variable keyboardList)
        'resize_keyboard' : true,           // Paramètre de mise en page
        'one_time_keyboard' : true          // Paramètre de mise en page
    }),
    disable_web_page_preview : true,         // Paramètre de mise en page
    parse_mode : "Markdown"                // Paramètre de mise en page
};
msg.payload.options = opts;                           // Introduction des options JSON
dans la charge utile

```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```
    }

    if (recv.toUpperCase() == (myDictKeyboard[myPCB][myMsg]).toUpperCase() &&
(global.get("PCBselected") == myPCB.toUpperCase())){ // Si j'ai cliqué
dans une option d'un menu d'un PCB (ex : "Set LED ON" du PCB MultiProtocol)
    msg.payload.content = "💬 I am in the *PCB " + myPCB + "* and I will
try to *" + myDictKeyboard[myPCB][myMsg] + "* ! Wait a moment..."; // Alors le bot
me répond
    global.set("MQTTrequest", (myPCB.toUpperCase()) + " " +
(myDictKeyboard[myPCB][myMsg]).toUpperCase()); // //
Création d'une chaîne permettant de créer la requête MQTT
    global.set("MQTTtopic", myPCB);
// Création du Topics MQTT qui contiendra la requête
}
}

return msg;
```

■ **Annexe 34 : NodeRed : Script Topics Payloads Manager**

```

msg.topic = global.get("MQTTtopic")+"/Request";
var myDictMQTTrequest = {
    "MULTIPROTOCOL GET DHT22 TEMPERATURE" : "temp",
    "MULTIPROTOCOL GET DHT22 HUMIDITY" : "hum",
    "MULTIPROTOCOL SET LED2 OFF" : "off",
    "MULTIPROTOCOL SET LED2 ON" : "on",
    "MONOPROTTOCOL ZIGBEE GET ELEGOO SENSOR A0" : "analog",
    "MONOPROTTOCOL ZIGBEE SET BUZZER ON" : "Bon",
    "MONOPROTTOCOL ZIGBEE SET BUZZER OFF" : "Boff",
    "MONOPROTTOCOL ZIGBEE SET LED2 ON" : "Lon",
    "MONOPROTTOCOL ZIGBEE SET LED2 OFF" : "Loff",
    "MONOPROTTOCOL LORA GET DS18B20 TEMPERATURE" : "temp",
    "MONOPROTTOCOL LORA GET LDR BRIGHTNESS" : "lum",
    "MONOPROTTOCOL LORA SET LED2 ON": "on",
    "MONOPROTTOCOL LORA SET LED2 OFF": "off"
};

for (var key in myDictMQTTrequest) {
    if (global.get("MQTTrequest") == key) {
        msg.payload = myDictMQTTrequest[key];
        global.set("currentPayload", myDictMQTTrequest[key]);
        global.set("MQTTrequest", "EMPTY");
        return msg;
    }
}

```

■ Annexe 35 : ESP32 TP Industry 4.0 : Script MultiProtocol Node

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include "DHT.h"
#include "SerialTransfer.h"
#include <SPI.h>
#include <LoRa.h>

//Configutation frd PIN : 3AU board LoRa
#define ss 0
#define rst 14
#define dio0 2
int counter = 0;
int myClock;
int myOldClock;
String LoRaMessageReceived;

//MQTT
const char* ssid = "Davide";
const char* password = "12345678";
const char* mqtt_server = "192.168.137.211";
WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg;
char msg[50];
int value;

//local datas
#define DHTTYPE DHT22
#define DHTPIN 27
DHT dht(DHTPIN, DHTTYPE); //déclaration du capteur
float temperature;
float humidity;
const int LED2 = 12;
const int BP1 = 36;
int presentBP1;
int pastBP1;

//XBee
SerialTransfer myTransfer;
struct STRUCT {
    int ESensorA0;
    int Buzzer;
    bool LED2;
    bool BP2;
} myXBeeBuffer;

int presentXBeeBP2;
int pastXBeeBP2;

void setup() {
    Serial.begin(115200);
    Serial2.begin(9600);
}
```

```

setup_wifi();
client.setServer(mqtt_server, 1883);
client.setCallback(callback);

pinMode(LED2, OUTPUT);
pinMode(BP1, INPUT);
dht.begin();

myTransfer.begin(Serial2);
LoRa.setPins(ss, rst, dio0);
while (!LoRa.begin(866E6)) {
    Serial.println(".");
    delay(500);
}
LoRa.setSyncWord(0xF3);
Serial.println("LoRa Initializing OK!");
}

void ReceiveLoRa() {

    int packetSize = LoRa.parsePacket();
    if (packetSize) {
        Serial.print("Received packet : ");
        while (LoRa.available()) {
            LoRaMessageReceived = LoRa.readString();
            Serial.println(LoRaMessageReceived);
        }
        if(LoRaMessageReceived == "BP2"){
            client.publish("MonoProtocol LoRa/Event", "*BP2* in the *PCB MonoProtocol
LoRa* is pushed !");
        }
        else {
            client.publish("MonoProtocol LoRa/RequestExecuted",
LoRaMessageReceived.c_str()); //accusé de réception
        }
    }
}

void SendLoRa(char *payload){
    LoRa.beginPacket();
    LoRa.print(payload);
    LoRa.endPacket();
}

void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
}

```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```

        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String payload;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        payload += (char)message[i];
    }
    Serial.println();
//-----MultiProtocol Requests-----
    if (String(topic) == "MultiProtocol/Request") {
        Serial.print("Changing output to ");
        if(payload == "on"){
            Serial.println("on");
            digitalWrite(LED2, HIGH);
            client.publish("MultiProtocol/RequestExecuted", "*LED2* : ON"); //accusé de
réception
        }
        else if(payload == "off"){
            Serial.println("off");
            digitalWrite(LED2, LOW);
            client.publish("MultiProtocol/RequestExecuted", "*LED2* : OFF"); //accusé de
réception
        }
        else if(payload == "temp"){
            temperature = dht.readTemperature();
            char tempString[30];
            sprintf(tempString,"*Temperature* : %0.2f°C", temperature);
            Serial.print("Temperature: ");
            Serial.println(tempString);
            client.publish("MultiProtocol/RequestExecuted", tempString); //accusé de
réception
        }
        else if(payload == "hum"){
            humidity = dht.readHumidity();
            char humString[30];
            sprintf(humString, "*Humidity* : %0.2f/100", humidity);
            Serial.print("Humidity: ");
            Serial.println(humString);
            client.publish("MultiProtocol/RequestExecuted", humString); //accusé de
réception
        }
    }
//-----MonoProtocol Zigbee Requests-----
    else if (String(topic) == "MonoProtocol Zigbee/Request") {

```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```

char RequestExecutedString[30];
uint16_t sendSize = 0;
if(payload == "analog"){
    sprintf(RequestExecutedString, "*Elegoo Sensor A0* : %u",
myXBeeBuffer.ESensorA0);
    client.publish("MonoProtocol Zigbee/RequestExecuted", RequestExecutedString);
//accusé de réception
}
else if(payload == "Bon"){
    myXBeeBuffer.Buzzer = 1;
    sendSize = myTransfer.txObj(myXBeeBuffer, sendSize);
    myTransfer.sendData(sendSize);
    sprintf(RequestExecutedString, "*Buzzer* : ON");
    client.publish("MonoProtocol Zigbee/RequestExecuted", RequestExecutedString);
//accusé de réception
}
else if(payload == "Boff"){
    myXBeeBuffer.Buzzer = 0;
    sendSize = myTransfer.txObj(myXBeeBuffer, sendSize);
    myTransfer.sendData(sendSize);
    sprintf(RequestExecutedString, "*Buzzer* : OFF");
    client.publish("MonoProtocol Zigbee/RequestExecuted", RequestExecutedString);
//accusé de réception
}
else if(payload == "Lon"){
    myXBeeBuffer.LED2 = 1;
    sendSize = myTransfer.txObj(myXBeeBuffer, sendSize);
    myTransfer.sendData(sendSize);
    sprintf(RequestExecutedString, "*LED2* : ON");
    client.publish("MonoProtocol Zigbee/RequestExecuted", RequestExecutedString);
//accusé de réception
}
else if(payload == "Loff"){
    myXBeeBuffer.LED2 = 0;
    sendSize = myTransfer.txObj(myXBeeBuffer, sendSize);
    myTransfer.sendData(sendSize);
    sprintf(RequestExecutedString, "*LED2* : OFF");
    client.publish("MonoProtocol Zigbee/RequestExecuted", RequestExecutedString);
//accusé de réception
}
}
//-----MonoProtocol LoRa Requests-----//
else if ((String(topic) == "MonoProtocol LoRa/Request")){
    char RequestExecutedString[30];
    if(payload == "temp"){
        SendLoRa("temp");
    }
    else if(payload == "lum"){
        SendLoRa("lum");
    }
    else if(payload == "on"){
        SendLoRa("on");
    }
    else if(payload == "off"){
        SendLoRa("off");
    }
}

```

```

        }

    }

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP32Client")) {
            Serial.println("connected");
            // Subscribe
            client.subscribe("MultiProtocol/Request");
            client.subscribe("MonoProtocol Zigbee/Request");
            client.subscribe("MonoProtocol LoRa/Request");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    ReceiveLoRa();

    presentBP1 = digitalRead(BP1);
    if (presentBP1 == 1 and pastBP1 == 0) {
        client.publish("MultiProtocol/Event", "*BP1* in the *PCB MultiProtocol* is
pushed !");
    }
    pastBP1 = presentBP1;

    presentXBeeBP2 = myXBeeBuffer.BP2;
    if (presentXBeeBP2 == 1 and pastXBeeBP2 == 0) {
        client.publish("MonoProtocol Zigbee/Event", "*BP2* in the *PCB MonoProtocol
Zigbee* is pushed !");
    }
    pastXBeeBP2 = presentXBeeBP2;

    if(myTransfer.available())
    {
        uint16_t recSize = 0;
        recSize = myTransfer.rxObj(myXBeeBuffer, recSize);
    }

    delay(20);
}

```

■ Annexe 36 : ESP32 TP Industry 4.0 : Script MonoProtocol Zigbee Node

```
#include "SerialTransfer.h"

int myClock;
int myOldClock;
int ESensorA0pin = 39;
int BP2pin = 35;
int LED2pin = 32;
int Buzzerpin = 2;
//XBee
SerialTransfer myTransfer;
struct STRUCT {
    int ESensorA0;
    int Buzzer;
    bool LED2;
    bool BP2;
} myXBeeBuffer;

void setup() {
    Serial.begin(115200);
    Serial2.begin(9600);
    myTransfer.begin(Serial2);
    pinMode(BP2pin, INPUT);
    pinMode(LED2pin, OUTPUT);

    ledcSetup(0, 2000, 8);
    ledcAttachPin(Buzzerpin, 0);
}

void loop() {
    if (myTransfer.available()) {
        uint16_t recSize = 0;
        recSize = myTransfer.rxObj(myXBeeBuffer, recSize);
        digitalWrite(LED2pin, myXBeeBuffer.LED2);
        ledcWrite(0, myXBeeBuffer.Buzzer*255);
        Serial.println("LED2 : " + String(myXBeeBuffer.LED2) + " ; Buzzer : " +
String(myXBeeBuffer.Buzzer));
    }

    myClock = (millis()/200)%2;
    if ((myClock == 1) and (myOldClock == 0)) {
        myXBeeBuffer.ESensorA0 = analogRead(ESensorA0pin);
        myXBeeBuffer.BP2 = digitalRead(BP2pin);
        uint16_t sendSize = 0;
        sendSize = myTransfer.txObj(myXBeeBuffer, sendSize);
        myTransfer.sendData(sendSize);
        //Serial.println(myXBeeBuffer.BP2);
    }
    myOldClock = myClock;
}
```

■ Annexe 37 : ESP32 TP Industry 4.0 : Script MonoProtocol LoRa Node

```
#include <SPI.h>
#include <LoRa.h>
#include <OneWire.h>
#include <DallasTemperature.h>

//Configutatation frd PIN : 3AU board multi-fontion
#define ss 0
#define rst 14
#define dio0 2

int TEMPpin = 25; // DS18B20 Temperature Sensor 1 Wire
char charTEMP[15];
OneWire oneWire(TEMPpin);
DallasTemperature sensors(&oneWire);

int LDRpin = 39;
char charLDR[15];
char charBP2[10];
int BP2pin = 35;
int LED2pin = 32;
char charLED2[15];
int myClock;
int myOldClock;
int counter;
int presentBP2;
int pastBP2;
void setup() {
    //initialize Serial Monitor
    Serial.begin(115200);
    while (!Serial) {
        Serial.println("LoRa Receiver");
        pinMode(LED2pin, OUTPUT);
        pinMode(BP2pin, INPUT);
        //setup LoRa transceiver module
        LoRa.setPins(ss, rst, dio0);

        //replace the LoRa.begin(---E-) argument with your location's frequency
        //433E6 for Asia
        //866E6 for Europe
        //915E6 for North America
        while (!LoRa.begin(866E6)) {
            Serial.println(".");
            delay(500);
        }
        // Change sync word (0xF3) to match the receiver
        // The sync word assures you don't get LoRa messages from other LoRa
        transceivers
        // ranges from 0-0xFF
        LoRa.setSyncWord(0xF3);
        Serial.println("LoRa Initializing OK!");
        sensors.begin();
    }
}
```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```

void SendLoRa(char *payload) {
    LoRa.beginPacket();
    LoRa.print(payload);
    LoRa.endPacket();
}

void loop() {
    // try to parse packet
    String LoRaData;
    int packetSize = LoRa.parsePacket();
    if (packetSize) {
        Serial.print("Received packet : ");
        while (LoRa.available()) {
            LoRaData = LoRa.readString();
            Serial.print(LoRaData);
        }
        if (LoRaData == "temp") {
            sprintf(charTEMP, "Temperature : %0.2f°C",
sensors.getTempCByIndex(0));
            //sprintf(charTEMP, "Temp : %u°C", counter);
            SendLoRa(charTEMP);
            Serial.println("Sending Packet: " + String(charTEMP));
            //counter++;
        }
        else if (LoRaData == "lum") {
            sprintf(charLDR, "Brightness : %u", analogRead(LDRpin));
            SendLoRa(charLDR);
            Serial.println("Sending Packet: " + String(charLDR));
        }
        else if (LoRaData == "on") {
            sprintf(charLED2, "LED2 : ON");
            SendLoRa(charLED2);
            digitalWrite(LED2pin, HIGH);
            Serial.println("Sending Packet: " + String(charLED2));
        }
        else if (LoRaData == "off") {
            sprintf(charLED2, "LED2 : OFF");
            SendLoRa(charLED2);
            digitalWrite(LED2pin, LOW);
            Serial.println("Sending Packet: " + String(charLED2));
        }
    }
}

// myClock = (millis()/1000)%2;
// if ((myClock == 1) and (myOldClock == 0)){
//     sensors.requestTemperatures();
//     sprintf(charLDR, "LIGHT : %u", analogRead(LDRpin));
//     sprintf(charTEMP, "TEMP : %0.2f", sensors.getTempCByIndex(0));
//     sprintf(charBP2, "BP2: %lu", digitalRead(BP2pin));
//     Serial.println(charLDR);
//     Serial.println(charTEMP);
//     Serial.println(charBP2);
// }
// myOldClock = myClock;

presentBP2 = digitalRead(BP2pin);

```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0

```
if (presentBP2 == 1 and pastBP2 == 0) {  
    SendLoRa("BP2");  
    Serial.println("BP2 pushed !");  
}  
pastBP2 = presentBP2;  
}
```

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**■ Annexe 38 : Firebase : Base de données en temps réel**

The screenshot shows the Firebase Realtime Database interface. At the top, there's a navigation bar with 'ISATfirebase' and a dropdown arrow. Below it, the main menu includes 'Realtime Database' (which is selected), 'Firestore Database', 'Storage', 'Hosting', 'Functions', and 'Machine Learning'. On the left, a sidebar lists 'Créer' (Create) and 'Publier et surveiller' (Publish and monitor). Under 'Publier et surveiller', there are sections for 'Crashlytics', 'Performance', 'Test Lab...', 'Analytics' (with sub-options: 'dashboard', 'Realtime', 'Events', 'Conversions', 'Audiences', 'Custom Definitions', 'Latest Release', 'DebugView'), and 'Spark' (with sub-options: 'Sans frais 0 \$/mois' and 'Changer de formule').

The main content area displays a log entry:

```

Protégez vos ressources Realtme Database des utilisations abusives telles que la fraude à la facturation et le hameçonnage
Configure App Check X
Accéder à la documentation D
?
```

Event: "2022-05-18 12:49:41 : *BP2* in the *PCB MonoProtocol LoRa* is pushed!"

Logs

- MonoProtocol LoRa
 - lum: "2022-05-18 12:44:33 : 373"
 - off: "2022-05-18 12:44:31 : OFF"
 - on: "2022-05-18 12:44:30 : ON"
 - temp: "2022-05-18 12:44:11 : 35.00°C"
- Zigbee
 - Borff: "2022-05-18 12:44:32 : OFF"
 - Bon: "2022-05-18 12:25:26 : ON"
 - Loff: "2022-05-18 12:44:30 : OFF"
 - Lon: "2022-05-18 12:44:29 : ON"
 - analog: "2022-05-18 12:44:34 : 2031"
- MultiProtocol
 - hum: "2022-05-18 12:49:25 : 37.60/100"
 - off: "2022-05-18 12:44:27 : OFF"
 - on: "2022-05-18 12:44:28 : ON"
 - temp: "2022-05-18 12:49:26 : 31.00°C"

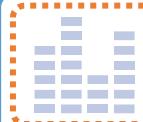
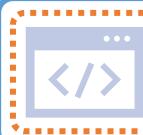
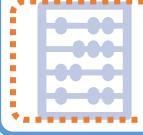
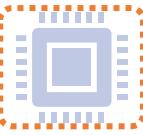
Emplacement de la base de données: Belgique (europe-west1)

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**■ Annexe 39 : Planification du projet**

La période de stage a duré 15 semaines. Etant donné que je l'ai effectuée en interne, je l'ai prolongé jusqu'à la fin de la remise du rapport (1^{er} juin).

Les deux dernières semaines, je les ai consacrées pour finaliser la rédaction du TFE. J'en ai profité pour réajuster ce projet avec quelques détails importants.

L'image représentée ci-dessous reprend l'ensemble de ma planification de ce TFE :

	Semaine 1	<ul style="list-style-type: none"> • Analyse du cahier des charges & recherches docum. • Développement des premières idées de conception 		Semaine 9 (Pâques)	<ul style="list-style-type: none"> • Début de montage vidéo des documents du pré-jury • Documentation approfondie de l'approche d'un serveur avec un dongle Zigbee2MQTT
	Semaine 2	<ul style="list-style-type: none"> • Conception d'un schéma bloc illustrant le PCB • Dimensionnement et début de conception du PCB 		Semaine 10 (Pâques)	<ul style="list-style-type: none"> • Complétion des documents du pré-jury • Réparation de l'imprimante 3D MK3 à l'école • Premiers tests avec RaspBerry PI 4 & Zigbee2MQTT
	Semaine 3	<ul style="list-style-type: none"> • Finalisation de la conception du PCB • Mise au propre des librairies 3D de Fusion360 du projet 		Semaine 11	<ul style="list-style-type: none"> • Soudage des 3 autres PCB (LoRa, Sigfox & Zigbee) • Finalisation des tests de programmation des PCB et validation du bon fonctionnement de ces derniers
	Semaine 4 (Carnaval)	<ul style="list-style-type: none"> • Achat du PCB après vérification • Documentation & rassemblement des scripts C de chaque périphérique du PCB 		Semaine 12	<ul style="list-style-type: none"> • Suite des tests avec RaspBerry PI 4 & Zigbee2MQTT • Début de la conception d'un Node Red
	Semaine 5	<ul style="list-style-type: none"> • Hébergement des scripts dans un dépôt GitHub • Début de conception des 3 autres PCB (LoRa, Sigfox & Zigbee) 		Semaine 13	<ul style="list-style-type: none"> • Première approche d'une API avec Telegram • Mise au propre de schémas & tableaux pour le rapport
	Semaine 6	<ul style="list-style-type: none"> • Finalisation & achat des 3 autres PCB • Arrivée du premier PCB 		Semaine 14	<ul style="list-style-type: none"> • Finalisation de l'approche API avec Telegram sur NodeRed • Arrivée du PCB V2.0 & soudage + tests
	Semaine 7	<ul style="list-style-type: none"> • Soudage du premier PCB • Finalisation des tests de programmation du PCB et validation du bon fonctionnement de ce dernier 		Semaine 15	<ul style="list-style-type: none"> • Ajout d'une base de données ainsi qu'un dashboard sur NodeRed • Finalisation de la création du TP Industrie 4.0
	Semaine 8	<ul style="list-style-type: none"> • Programmation des modules radios et validation du bon fonctionnement de ces derniers • Conception d'une V2.0 du premier PCB 			

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**■ Annexe 40 : Liste de matériel : PCB Multi-Protocole V2.0**

NOTES	COMPOSANT	QUANTITÉ POUR UN PCB	PRIX À L'UNITÉ
	capacitor 1000µF 10V	1	0.145€/u
	capacitor 10µF 25V	3	0.0759€/u
	Xbee (protocole zigbee)	1	39.23€/u
	Potentiometer 10kOhm	1	0.501€/u
	Connecteur SMT IPEX (antenne)	1	1€/u
POUR CAPTEUR PIR	Header female 2.54mm 3pin	1	0.143€/u
POUR MPU6050 & SIGFOX	Header female 2.54mm 8pin	3	1.23€/u
POUR SIGFOX & MICROPHONE	Header female 2.54mm 4pin	2	0.144€/u
	Header male 2mm 8pins	2	0.354€/u
POUR L'ESP32	Header female 2.54mm 20pin	2	0.516€/u
	Régulateure de tension	1	1€/u
	MCP23008	1	1.54€/u
	DHT22	1	10.37/u
	Résistances 10kOhm 1/4W	6	0.0594€/u
	Résistances 20kOhm 1/4W	1	0.0594€/u
	Résistances 330kOhm 1/4W	8	0.0594€/u
	capacitor 330pF	14	0.197€/u
	Jumper/cavalier pack de 600	0.00333	4.19€/600u
	MCP23008 socket/shield	1	0.614€/u
	Entretroise M3 10mm	4	0.185€/u
	TSOP18638 (IR receiver)	1	0.54€/u
	Régulateure de tension	1	1€/u
CAN	Résistances 120Ohm	1	0.0594€/u
	MCP2562 socket/shield	1	0.286€/u
	Diode TVS	1	0.723€/u
	Bornier	1	0.322€/u
	boutons pousoirs	0.01	10€/200u
	LEDs	0.00∞	10€/500u
	header male (2.54mm)	0.00∞	10.79€/60u (60*40pins)
	SONOFF ZigBee 3.0 USB Dongle Plus	NA	28,99€/u
	LED RGB NeoPixel	3	0.289€/u
ANTENNE COMPRISE	BRKWS01 (SigFox) (France)	1	23.88€/u
	RFM95 (LoRa) Chez Mouser	1	7.4€/u
	RFM95 (LoRa) Chez Farnell	1	26.86€/u
	IR LED	1	0.095€/u
	Header female 2mm 8pins	2	1.506€/u
	Header female 2mm 10pin	2	1.622€/u

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**■ Annexe 41 : Liste de matériel : PCB Mono-Protocole orienté LoRa V1.1**

NOTES	COMPOSANT	QUANTITÉ POUR UN PCB	PRIX À L'UNITÉ
	capacitor 10µF 25V	3	0.0759€/u
	Connecteur SMT IPEX (antenne)	1	1€/u
MIKROBUS	Header female 2.54mm 8pin	2	1.23€/u
OLED	Header female 2.54mm 4pin	1	0.144€/u
POUR L'ESP32	Header female 2.54mm 20pin	2	0.516€/u
	Résistances 10kOhm 1/4W	6	0.0594€/u
CAN	Résistances 120Ohm	1	0.0594€/u
LEDS	Résistances 330Ohm 1/4W	5	0.0594€/u
	capacitor 0.1µF	11	0.17€/u
	MCP2562 socket/shield	1	0.286€/u
	MCP23008	1	1.54€/u
	MCP23008 socket/shield	1	0.614€/u
	Entretroise M3 10mm	4	0.185€/u
	TSOP18638 (IR receiver)	1	0.54€/u
LORA	Header male 2mm 8pins	2	0.354€/u
	Diode TVS	1	0.723€/u
	Bornier	1	0.322€/u
	Régulateure de tension	1	1€/u
	boutons poussoirs	0.01	10€/200u
	LEDs	0.00∞	10€/500u
	header male (2.54mm)	0.00∞	10.79€/60u (60*40pins)
	LDR (GL5516 : 5k - 10k)	0.01	8€/100u
	RFM95 (LoRa) Chez Mouser	1	7.4€/u
	RFM95 (LoRa) Chez Farnell	1	26.86€/u
	IR LED	1	0.095€/u
	MCP2562	1	1.08€/u
LORA	Header female 2mm 8pins	2	1.506€/u
	DS2812B	1	5.552€/u

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**■ Annexe 42 : Liste de matériel : PCB Mono-Protocole orienté SigFox V1.1**

NOTES	COMPOSANT	QUANTITÉ POUR UN PCB	PRIX À L'UNITÉ
SERVO MOTOR	capacitor 10µF 25V	3	0.0759€/u
MIKROBUS & SIGFOX	Header female 2.54mm 3pin	1	0.143€/u
OLED & SIGFOX & SONAR & SENSOR ELEGOO	Header female 2.54mm 8pin	4	1.23€/u
POUR L'ESP32	Header female 2.54mm 4pin	4	0.144€/u
SONAR	Header female 2.54mm 20pin	2	0.516€/u
	Résistances 20kOhm 1/4W	1	0.0594€/u
	Résistances 10kOhm 1/4W	5	0.0594€/u
CAN	Résistances 120Ohm	1	0.0594€/u
LEDS	Résistances 330Ohm 1/4W	8	0.0594€/u
	capacitor 0.1µF	12	0.17€/u
	MCP2562 socket/shield	1	0.286€/u
	MCP23008	1	1.54€/u
	MCP23008 socket/shield	1	0.614€/u
	Entretoise M3 10mm	4	0.185€/u
	Diode TVS	1	0.723€/u
	Bornier	1	0.322€/u
	Régulateure de tension	1	1€/u
	boutons pousoirs	0.01	10€/200u
	LEDs	0.00∞	10€/500u
	header male (2.54mm)	0.00∞	10.79€/60u (60*40pins)
ANTENNE COMPRISE	BRKWS01 (SigFox) (France)	1	23.88€/u
	MCP2562	1	1.08€/u

CARTES 3AU PROTOTYPAGE INDUSTRIE 4.0**■ Annexe 43 : Liste de matériel : PCB Mono-Protocole orienté Zigbee V1.1**

NOTES	COMPOSANT	QUANTITÉ POUR UN PCB	PRIX À L'UNITÉ
	Xbee module	1	39.23€/u
	capacitor 10µF 25V	3	0.0759€/u
MIKROBUS	Header female 2.54mm 8pin	2	1.23€/u
OLED & SENSOR ELEGOO	Header female 2.54mm 4pin	2	0.144€/u
POUR L'ESP32	Header female 2.54mm 20pin	2	0.516€/u
	Résistances 10kOhm 1/4W	5	0.0594€/u
CAN	Résistances 120Ohm	1	0.0594€/u
LEDS & TOUCH BUTTON	Résistances 330Ohm 1/4W	5	0.0594€/u
	capacitor 0.1µF	11	0.17€/u
	MCP2562 socket/shield	1	0.286€/u
	MCP23008	1	1.54€/u
	MCP23008 socket/shield	1	0.614€/u
	Entretroise M3 10mm	4	0.185€/u
	Diode TVS	1	0.723€/u
	Bornier	1	0.322€/u
XBEE	Header male 2mm 8pins	2	0.354€/u
	Régulateure de tension	1	1€/u
	Buzzer	1	0.399€/u
	2N70	1	0.552€/u
	boutons pousoirs	0.01	10€/200u
	LEDs	0.00∞	10€/500u
	header male (2.54mm)	0.00∞	10.79€/60u (60*40pins)
	LED RGB NeoPixel	3	0.289€/u
	MCP2562	1	1.08€/u
XBEE	Header female 2mm 8pins	2	1.506€/u

CONCEPTION DE CARTES ELECTRONIQUES POUR L'INDUSTRIE 4.0

COMMUNICATIONS SANS-FIL (LORA, ZIGBEE, SIGFOX, WI-FI, ...)

Pour simuler de façon didactique le principe de l'Industrie 4.0 et de l'IoT, plusieurs cartes électroniques ont été conçues.

Ces cartes embarquent toutes un microcontrôleur commun, l'ESP32. Il centralise plusieurs périphériques, notamment des capteurs/actionneurs, des émetteurs/récepteurs radio, etc.

Comme en entreprise, la communication entre machines (M2M), l'accès aux données depuis des serveurs, et les interfaces utilisateurs s'effectuent par le biais de plusieurs outils & protocoles.

Parmi ces outils, se trouvent initialement des programmes IDE, des services API, des Clouds, des passerelles, des bases de données, des fournisseurs de réseaux IoT (SigFox, LoRaWAn, ...), etc.

Du coté des protocoles de communication, les cartes conçues dans ce TFE sont compatibles avec Wi-Fi, Bluetooth, Zigbee, LoRa, SigFox, CAN, IR, I²C & SPI.

Ainsi, en assemblant toute une étude de simulation d'un processus IoT à partir de ces outils & protocoles, il a été possible de réaliser un système assez proche d'une application typique de l'Industrie 4.0.

En somme, l'émulation de ce processus sous les cartes électroniques de ce TP aborde une étendue immense de plusieurs aspects IoT, notamment le principe d'émetteur/récepteur radio, le courtier MQTT, les requêtes HTTP, les formats JSON, etc.



Par l'étudiant DI VENTI Davide

Haute Ecole EPHEC-ISAT – Département Technique

<https://github.com/DavideDiVenti/TFE-3AU-2022>

