UNIVERSITÉ TOULOUSE III PAUL SABATIER

Faculté Sciences et Ingénierie

Université de Toulouse

Projet long: CAN bus over DC network for Raspberry pi

Environnement : NeoCampus

MASTER 2 : Systèmes informatiques, ambiants, mobiles et embarqués

Réalisé par : ZAKRI Imane | ALAMI Zouhir | SENSOU Roumayssae

Encadré par : Dr. François. Thiebolt

2021/2022

Table des matières

Table de figures

Introduction

NeOCampus est un terrain d'expérimentations et d'innovations localisé sur le campus science de Rangueil, Université Toulouse III Paul Sabatier, ouvert à tous les laboratoires de recherche de la Région et aux partenariats avec des industriels. Les principaux axes de recherche concernent l'énergie, l'eau et l'air, la qualité de vie à l'extérieur et à l'intérieur des bâtiments, la biodiversité, le développement durable, la mobilité et l'éco-citoyenneté.[1]

Notre projet se déroule dans le contexte des systèmes ambiants appliqués dans le cadre du neoCampus, visant à faciliter la vie des utilisateurs au sein du campus, en s'appuyant sur la mise en place d'un bus CAN qui gère la communication entre une carte STM32(liée aux capteurs/actionneurs) et la carte Raspberry Pi auquel va être raccordé des modules d'entrées / sorties CAN et qui gère le coté logiciel (DomoticZ).

Notre projet se divise sur quatre parties, nous avons travaillé sur l'acquisition des données des capteurs (luminosité, température) dans la carte STM32, la deuxième partie concerne la configuration et la mise en œuvre de la communication CAN entre les deux cartes embarqués STM32 et Raspberry pi. La troisième partie, s'intéresse au côté logiciel du projet sur la plateforme Domoticz qui donne une visibilité des données acquises et contrôle en contrepartie la lumière (exemple : des salles). La quatrième et la dernière partie concerne l'implémentation du protocole « sigle wire CAN ».

Objectifs

- Matériel : Test Bench basé sur RPi et module CAN.
- Système : configuration RPi OS pour contrôler CAN MCP2515 via SPI, et ajout d'une bibliothèque pour ce propos.
- PCB : Création d'un PCB composé d'un STM32F103c8t6, et d'un transceiver CAN.
- Home Assistant : configurez le PCB pour communiquer avec Home Assistant.
- Modulation : implémentez la norme SWCAN pour que le CAN soit piloté directement avec la ligne d'alimentation 24 DC.

Etude théorique du projet

I. Protocoles de communication

I.1. Protocole CAN

I.1.1 Origines et utilisations du réseau CAN

Pour satisfaire des exigences de plus en plus importantes du client en matière de sécurité et de confort, et pour se conformer aux lois de réduction de la pollution et de la consommation de plus en plus drastiques, l'industrie automobile a développé de nombreux systèmes électroniques : systèmes anti-patinage, contrôle électronique du moteur et de l'air climatisé, fermeture centralisée des portes, etc. La complexité de ces systèmes et la nécessité d'échanger des données entre eux signifient de plus en plus de câbles. A côté du coût très important de ce câblage, la place qui lui est nécessaire pouvait le rendre tout simplement impossible à installer. Enfin, le nombre croissant de connexions et de câbles posait de sérieux problèmes de fiabilité et de réparation. La société Robert Bosch GmbH (Allemagne), un important équipementier automobile, a fourni une solution dans les années 1980 avec le bus CAN (Controller Area Network). L'entreprise allemande a défini le protocole et a autorisé de nombreux autres fabricants à développer des composants compatibles CAN. CAN est pensé et réalisé pour répondre à des impératifs de robustesse, de fiabilité, de simplicité et d'économie liés aux productions de masse de l'industrie automobile. CAN possède donc toutes les qualités pour séduire beaucoup d'industriels, soucieux de retrouver dans leurs installations ou leurs équipements, la fiabilité, la robustesse et le faible coût d'un système de communication normalisé et éprouvé. CAN est un réseau de communication série qui supporte efficacement le contrôle en temps réel de systèmes distribués tels qu'on peut en trouver dans les automobiles, et ceci avec un très haut niveau d'intégrité au niveau des données. Avec le protocole CAN, les contrôleurs, capteurs et actionneurs communiquent entres eux à une vitesse pouvant aller jusqu'à 1 Mbits/s. CAN est utilisé surtout pour la mise en réseau des organes de commande du moteur, de la boîte à vitesse, de la suspension et des freins (figure 1). Il s'agit là d'applications temps réel et critiques. Pour la mise en réseau des organes dits de carrosserie et de confort (commande des feux, des lève-vitres, de la climatisation, du verrouillage central, réglage de sièges et de rétroviseur), les constructeurs peuvent faire appel à CAN ou à d'autres réseaux de terrain comme VAN (Vehicle Area Network).

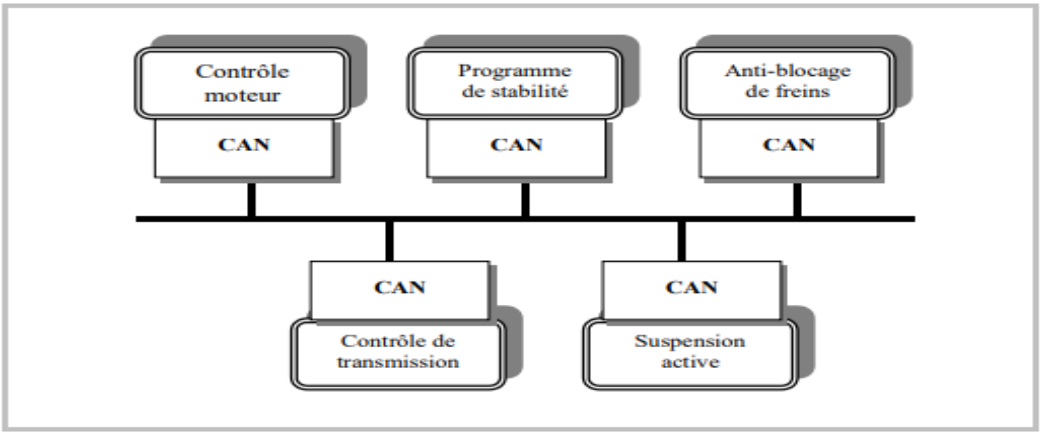

FIGURE 1: CAN DANS L'AUTOMOBILE

I.1.2. CAN dans le modèle ISO/OSI

CAN est un réseau compatible au modèle de référence ISO/OSI (ISO : International Organization for Standards, OSI : Open Systems Interconnection). CAN a été normalisé par l'ISO dans les normes 11898 pour les applications à haute vitesse (jusqu'à 1 Mb/s) et ISO 11519 pour les applications à basse vitesse (jusqu'à 125 kb/s). Comme le montre la figure 2, CAN (à l'image de la quasi-totalité des réseaux locaux industriels) a une architecture en trois couches ; les couches 3 à 6 du modèle OSI sont vides dans les architectures fondées sur CAN. Les spécifications CAN s'intéressent essentiellement aux couches MAC et physique.

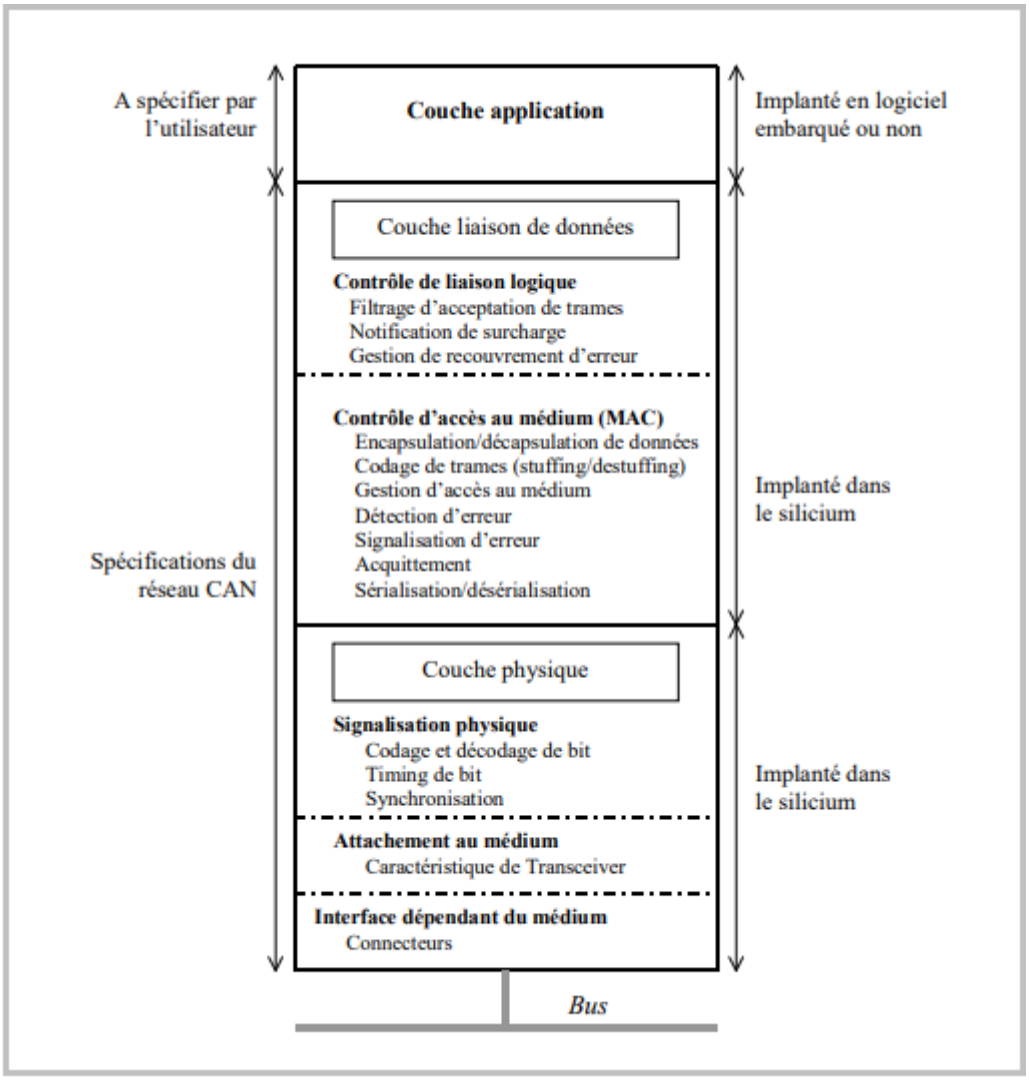

FIGURE 2: ARCHITECTURE CAN.

I.1.3. Fonctionnement du réseau CAN

1) Couche physique

Généralement, CAN utilise comme support de transmission une paire torsadée blindée ou non blindée. Les nœuds sont reliés entre eux par l'intermédiaire d'un bus série équipé de terminateurs de lignes (résistances de terminaison).

Les interfaces physiques sont de type différentiel en mode tension, proche du principe de la liaison RS485. La figure 3 montre un exemple de raccordement au bus CAN.

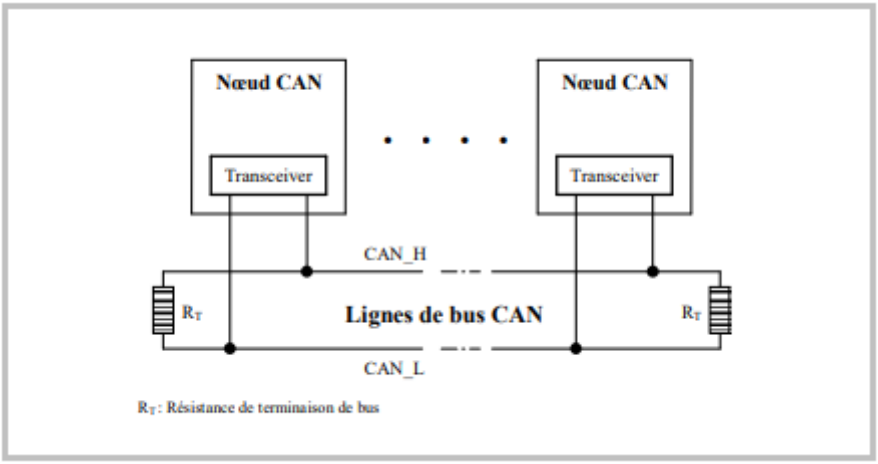

FIGURE 3: TOPOLOGIE DU BUS CAN

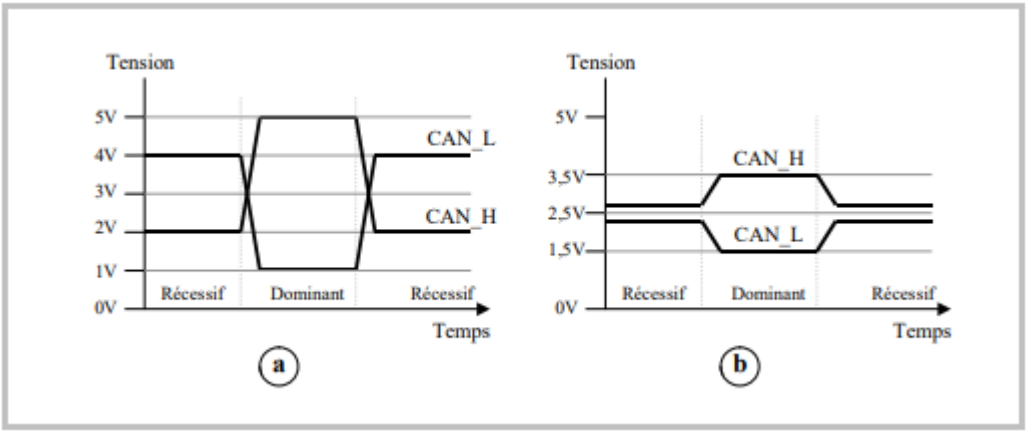

FIGURE 4: REPRÉSENTATION PHYSIQUE DES BITS

a) cas de la norme ISO 11519 (basse vitesse)
b) cas de la norme ISO 11898 (haute vitesse)

Parmi les multiples techniques de codage existantes (NRZ, NRZI, Manchester simple, Manchester différentiel, etc.), CAN a retenu le code NRZ (Non Return to Zero) pour sa simplicité. Avec le code NRZ, la valeur du signal reste constante pendant toute la durée d'un bit. Dans la norme ISO 11898 (haute vitesse), les nœuds détectent un bit récessif si la différence de tension entre les lignes CAN_H et CAN_L est inférieure ou égale à 0,5 V. Ils détectent un bit dominant si cette différence est supérieure ou égale à 0,9 V. La tension nominale pour le bit dominant est 3,5 V pour la ligne CAN_H et 1,5 V pour la ligne CAN_L. Dans la norme ISO 11519 (basse vitesse), les nœuds détectent un bit récessif si la différence de tension entre CAN_L et CAN_H est de l'ordre de 1,5 V. Ils détectent un bit dominant si cette différence est de l'ordre de - 3 V.

2) Formats de trames de CAN

Le protocole CAN 2.0 comporte deux spécifications qui diffèrent uniquement au niveau de la longueur de l'identificateur.

La version 2.0A définit des identificateurs de 11 bits (on parle dans ce cas de trame de format standard) et la version 2.0B des identificateurs de 29 bits (on parle dans ce cas de trame de format étendu).

Pour permettre le développement de contrôleurs assez simples, le support complet du format étendu n'est pas requis pour être conforme au protocole 2.0.

Les contrôleurs sont considérés conformes au protocole 2.0 s'ils respectent les deux conditions suivantes :

• Le format standard doit être totalement supporté.

• Ils doivent être capables de recevoir des trames de format étendu, mais sans forcément être capables de les traiter ; elles ne doivent seulement pas être détruites.

3) Types de trames

Il existe quatre types de trames pouvant être transmises sur un bus CAN :

• Trames de données : elles sont utilisées pour transporter des données (messages) de l'application (ou valeurs d'objets) sur le bus. C'est le producteur d'un identificateur qui émet des trames de données associées à cet identificateur.

• Trames de requête distante : elles sont utilisées par un nœud (un consommateur) pour demander la transmission de trames de données par d'autres nœuds (les producteurs) avec le même identificateur. Le bit RTR (Remote Transmission Request) permet de distinguer les trames de données des trames de requête. Le bit RTR est égal à 0 pour une trame de données et à 1 pour une trame de requête. On notera que les trames de données sont prioritaires par rapport aux trames de requête : quand un producteur et un consommateur d'un même objet entrent en conflit, c'est la trame émise par le producteur qui s'impose, ce qui est normal puisque la trame du producteur répond à la requête faite par le consommateur.

• Trames d'erreur : elles sont transmises par un nœud ayant détecté une erreur. Leur format et utilisation seront détaillés par la suite.

• Trame de surcharge : elles sont utilisées pour demander un délai entre deux trames de données ou de requête successives. Les trames de données ou de requête sont séparées des trames qui les précèdent (de quelque type qu'elles soient) par un temps dit intertrame (ce temps doit correspondre à au moins le temps de 3 bits). Les trames d'erreur ou de surcharge ne sont pas séparées des autres trames par un intertame.[2]

I.2. Protocole SPI

Une liaison SPI (Serial Peripheral Interface) est un bus de données série synchrone baptisé ainsi par Motorola, qui opère en mode Full-duplex. Les circuits communiquent selon un schéma maître esclaves, où le maître s'occupe totalement de la communication. Plusieurs esclaves peuvent coexister sur un même bus, dans ce cas, la sélection du destinataire se fait par une ligne dédiée entre le maître et l'esclave appelée chip select.

Dans les circuits d'interface le bus SPI n'est pas seul est se trouve talonné depuis de nombreuses années par les circuits dotés d'une liaison série synchrone de type SPI. Cette appellation rencontre cependant des signaux et des chronogrammes qui sont beaucoup moins bien normalisés que ceux du bus I2C, tout simplement parce que la liaison du bus SPI ne fait l'objet d'aucune norme officielle.

Certains circuits indiquent qu'ils sont munis d'une interface SPI, mais vous pourrez aussi trouver comme appellation « Microwire » (qui est une marque déposée de National Semiconductor) ou bien encore « bus série trois fils ».

Un bus de ce type permet la connexion, sous forme série maître esclave, de plusieurs circuits disposant d'interfaces compatibles, avec seulement trois fils de liaisons. Les dernières versions d'Arduino incluent une bibliothèque qui permet la communication avec les périphériques SPI. SPI a des lignes labellisés « MOSI » : Master Output Slave Input, généré par le maître, des lignes labellisées « MISO » : Master Input Slave Output, généré par l'esclave et une ligne d'horloge (SCK : SPI Serial Clock). Ces trois lignes sont connectées à leurs lignes respectives sur un ou plusieurs esclaves. Les esclaves sont identifiées grâce au signal de leur ligne SS (Slave Select). Autres nommages possibles : SDO (Serial Data Out) et SDI (Serial Data Input). La figure ci-contre illustre les connexions SPI.

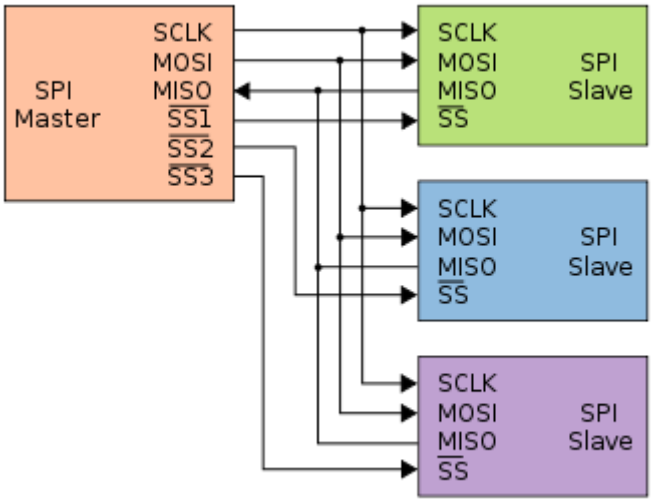

FIGURE 5: ARCHITECTURE SPI

Le maître sélectionne un seul et unique esclave avec lequel il veut rentrer en communication par la mise à niveau logique zéro de /SS 1 2 3, puis, après 8 fronts d'horloge, l'octet de donnée est transféré. La patte MISO de l'esclave non sélectionné est à l'état haute impédance. La seule limite au nombre d'esclaves est en fait la possibilité de broches SS du maître. Cas de la daisy chain (esclave en guirlande, en cascade...) Le maître sélectionne tous les esclaves par la mise à niveau logique zéro de /SS, puis après 3*8 fronts d'horloge, les 3 octets de données sont transférés (dans le cas d'un octet par esclave). Cette disposition permet de réduire le nombre de lignes /SS, mais en contrepartie il faudra un "buffer" plus grand dans le maître (ou une gestion du soft plus élaborée).[3]

I.3. Protocole I2C

Le bus I2C (Inter Integrated Circuit Bus) est le bus historique, développé par Philips pour les applications de domotique et d'électronique domestique au début des années 80, notamment pour permettre de relier facilement à un microprocesseur les différents circuits d'un téléviseur moderne.

C'est le bus qui a émergé de la guerre des standards lancée par tous les acteurs du monde électronique. Ainsi, dans votre téléviseur, tous les ensembles sont sur un bus I2C (Récepteur télécommande, réglages ampli BF, tuner, horloge, gestion péritel)

Il existe d'innombrables périphériques exploitant ce bus, il est même implantable par logiciel dans n'importe lequel des microcontrôleurs. Le poids de l'industrie de l'électronique grand public a imposé des prix très bas aux nombreux composants.

Le bus I2C permet de faire communiquer entre eux des composants électroniques très divers grâce à seulement trois fils :

- Un signal de donnée (SDA),
- Un signal d'horloge (SCL),
- Un signal de référence électrique (Masse).

Ceci permet de réaliser des équipements ayants des fonctionnalités très puissantes (En apportant toute la puissance des systèmes microprogrammés) et conservant un circuit imprimé très simple, par rapport un schéma classique (8bits de données, 16 bits d'adresse + les bits de contrôle).

Les données sont transmises en série à 100Kbits/s en mode standard et jusqu'à 400Kbits/s en mode rapide. Ce qui ouvre la porte de cette technologie à toutes les applications où la vitesse n'est pas primordiale.[4]

Principe :

Afin de d'éviter les conflits électriques les Entrées/Sorties SDA et SCL sont de type "Collecteur Ouvert". Cela permet ainsi la présence de plusieurs maîtres sur le bus.

Structure d'E/S d'un module I2C :

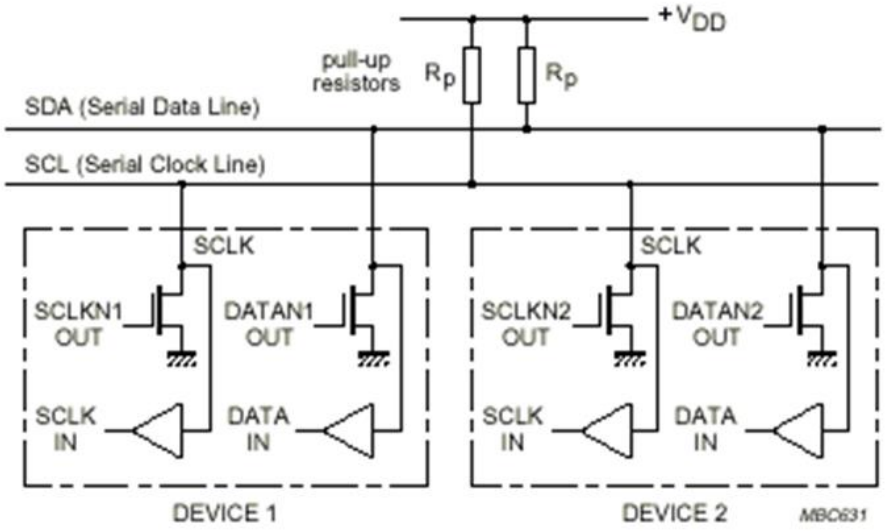

FIGURE 6: PRINCIPE DE BUS I2C

I.4. Protocole MQTT

MQTT, pour "Message Queuing Telemetry Transport", est un protocole open source de messagerie qui assure des communications non permanentes entre des appareils par le transport de leurs messages.

MQTT est un standard ISO (ISO/IEC PRF 20922). Au départ, TT (premier nom du protocole) a été développé par Andy Stanford-Clark (IBM) et Arlen Nipper (Eurotech, actuellement Cirrus Link) en 1999 pour le contrôle d'un pipeline dans le désert.

L'objectif était d'avoir un protocole de bande passante efficace utilisant peu d'énergie à un moment où les périphériques ne pouvaient être connectés qu'au travers de satellites. Le protocole MQTT utilise une architecture « publish/subscribe » en contraste avec le protocole HTTP et son architecture « request/response ».

Le point central de la communication est le broker MQTT en charge de relayer les messages des émetteurs vers les clients. Chaque client s'abonne via un message vers le broker : le « topic » (sorte d'information de routage pour le broker) qui permettra au broker de réémettre les messages reçus des producteurs de données vers les clients. Les clients et les producteurs n'ont ainsi pas à se connaître, ne communiquant qu'au travers des topics. Cette architecture permet des solutions multi-échelles.

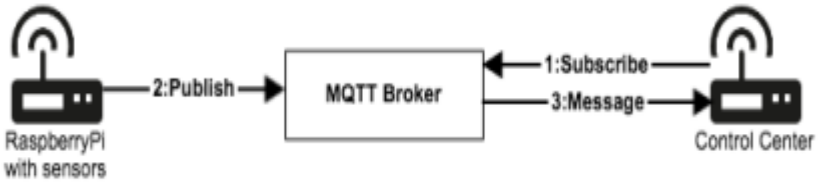

FIGURE 7: ARCHITECTURE MQTT

Chaque client MQTT a une connexion ouverte en permanence avec le broker. Si la connexion s'arrête, le broker bufférise les messages et les émet dès que la reconnexion avec le client est effectuée. Un « topic MQTT » est une chaîne de caractères qui peut posséder une hiérarchie de niveaux séparés par le caractère « / ». Par exemple, une information de température du salon pourrait être envoyée sur le topic « maison/salon/température » et la température de la cuisine sur « maison/cuisine/température ».

Le signe « + » est un caractère « wildcard » qui permet des valeurs arbitraires pour une hiérarchie particulière et le signe « # » pour plus d'un niveau. Les messages envoyés peuvent être de toutes sortes mais ne peuvent excéder une taille de 256 Mo.

MQTT intègre en natif la notion de QoS. En effet le publieur à la possibilité de définir la qualité de son message. Trois niveaux sont possibles :

• Un message de QoS niveau 0 « At most once » sera délivré tout au plus une fois. Ce qui signifie que le message est envoyé sans garantie de réception, (le broker n'informe pas l'expéditeur qu'il l'a reçu et le message)

• Un message de QoS niveau 1 « At least once » sera livré au moins une fois. Le client transmettra plusieurs fois s'il le faut jusqu'à ce que le Broker lui confirme qu'il a était transmis sur le réseau.

• Un message de QoS niveau 2 « exactly once » sera obligatoirement sauvegardé par l'émetteur et le transmettra toujours tant que le récepteur ne confirme pas son envoi sur le réseau. La principale différence étant que l'émetteur utilise une phase de reconnaissance plus sophistiquée avec le broker pour éviter une duplication des messages (plus lent mais plus sûr).[5]

II. Outils logiciels

II.1. DomoticZ

Domoticz est un logiciel open sources et gratuit de domotique, consommant peu de ressources système. Il fonctionne sur plusieurs OS (Linux, Windows). Tel un système domotique digne de ce nom il va vous permettre de configurer votre système domotique comme par exemple d'avoir des détecteurs d'ouverture, de mouvement, incendie, inondation…, mais aussi des capteurs de température, d'humidité, … de gérer des caméras IP, de gérer votre lumière… bref un système libre est complet. C'est aussi le moyen de créer son système domotique à bas prix.

L'autre avantage indéniable c'est qu'il ne nécessite pas de Cloud, en d'autres mots pas besoin d'hébergement ailleurs ![6]

FIGURE 8: LOGO DU LOGICIEL

II.2. Mosquitto

Mosquitto est le broker le plus souvent utilisé pour les projets ESP8266 (Arduino et Raspberry). Lancé en 2008, il est disponible sur toutes les plateformes (MacOS, Windows XP-10, Linux). Deux méthodes sont possibles pour l'installer : depuis le terminal d'un ordinateur (en connectant au préalable le clavier, la souris et le moniteur à un Pi), avant de le lancer depuis le terminal, ou (toujours par le biais d'un ordinateur) de l'installer en utilisant putty (SSH) et en accédant au Root.[7]

III. Outils matériels

III.1. STM32

La famille STM32 est une série de microcontrôleurs 32-bits en circuits intégrés réalisés par la société Franco-Italienne STMicroelectronics. Les puces STM32 sont regroupées dans différentes séries proches, basées sur les processeurs d'architecture ARM 32-bits, tels que le Cortex-M7F, le Cortex-M4F, le Cortex-M3, Cortex-M0+, ou le Cortex-M0.

Chaque microcontrôleur est constitué d'un cœur de calcul, de mémoire vive (RAM) statique, de mémoire flash (pour le stockage), d'une interface de débogage et de différents périphériques1.

Le STM32F103C6T6 est un microcontrôleur ARM Cortex-M3 32 bits de ligne de performance de densité moyenne en boîtier LQFP 48 broches. Il incorpore un coeur RISC haute performance avec une fréquence de fonctionnement 72MHz, des mémoires embarquées haute vitesse, une gamme étendue d'E/S optimisées et des périphériques connectés à deux bus APB. Le STM32F103C6T6 est doté d'un CAN 12 bits, de timers, d'un timer PWM, d'interfaces de communication avancées et standard. Un jeu complet de modes à économie d'énergie pour la conception d'applications faible consommation.[8]

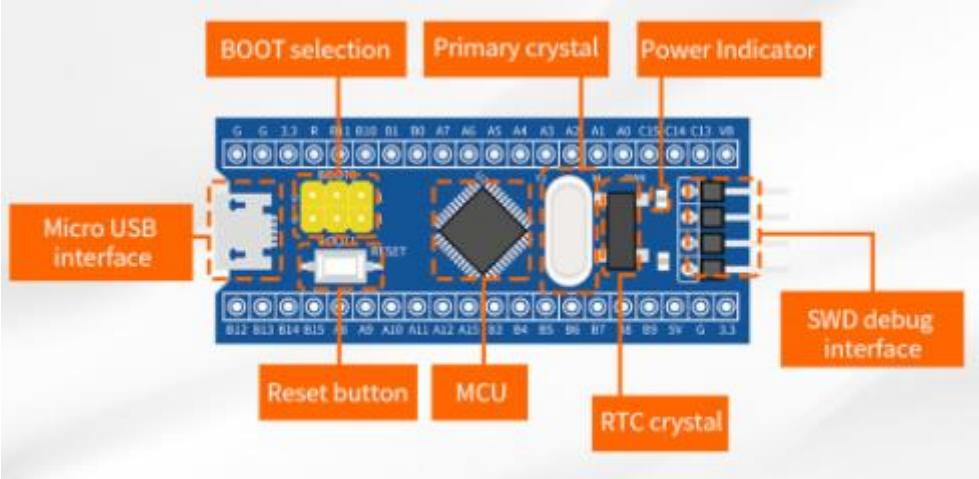

FIGURE 9: STM32F103C6T6

Caractéristiques techniques

- Plage de tension d'utilisation de 2V à 3.6V
- 64Ko de mémoire flash
- 20Ko de SRAM
- Unité de calcul CRC, ID unique 96 bits
- Deux convertisseurs A/N 1µs 12 bits (jusqu'à 10 voies)
- Contrôleur DMA 7 voies, 3 timers à usage général et 1 timer de contrôle avancé
- 37 ports E/S rapides
- Interfaces de débogage fil série (SWD) et JTAG
- Interfaces : deux SPI, deux I2C, trois USART, une USB et une CAN
- Gamme de température ambiante de -40°C à 85°C

III.2. Raspberry pi

La Raspberry Pi est ce que l'on appelle un nano-ordinateur monocarte. Plus simplement, on pourrait résumer la Raspberry Pi en disant que c'est un ordinateur de la taille d'une carte de crédit qui coûte environ 35€.

Très abordable en termes de prix, consommant peut d'énergie et disposant de connecteurs adaptés à électronique, la Raspberry Pi est donc très adaptée pour apprendre la programmation, faire de l'électronique, de la domotique, etc. [9]

FIGURE 10: RASPBERRY PI 3

III.3. MCP2515

Le CAN-BUS est un bus industriel courant en raison de sa grande portée filaire, de sa vitesse de communication moyenne et de sa grande fiabilité. On le trouve couramment sur les machines-outils modernes et comme bus de diagnostic automobile.

Ce module CAN-BUS donne à votre Arduino/Seeeduino la capacité CAN-BUS. Avec un câble convertisseur OBD-II ajouté et la bibliothèque OBD-II importée, vous êtes prêt à construire un dispositif de diagnostic embarqué ou un enregistreur de données.

Implémente CAN V2.0B à 1 Mb/s maximum

Interface SPI jusqu'à 10 MHz

Données standard (11 bits) et étendues (29 bits) et trames distantes

Deux tampons de réception avec stockage des messages prioritaires [10]

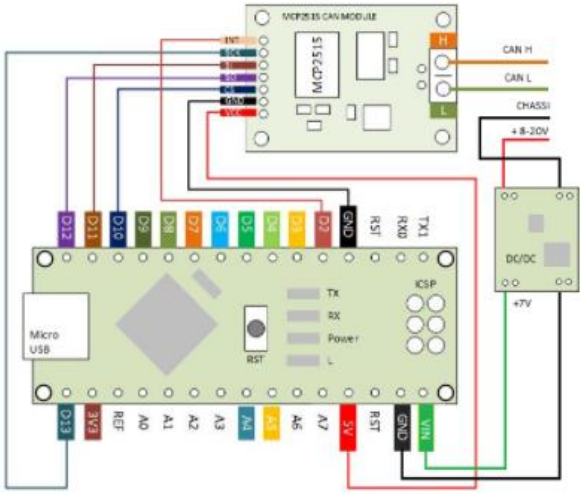

FIGURE 11: ARCHITECTURE MCP2515

Caractéristiques techniques :

- Support CAN V2.0 B, Taux de transfert jusqu'à 1Mb/S
- Alimentation électrique : DC 5V, Contrôle du protocole SPI
- Consommation au travail : 5mA
- Courant de veille : 1uA
- Taille : 4x2,8 cm,
- Température de fonctionnement : -40 C - +85 C

III.4. CAN Transceiver

Caractéristiques :

1. Environnement d'utilisation : ce Module peut être utilisé dans un environnement à haute interférence.

2. Conformité à la norme : l'appareil peut envoyer et recevoir des marchandises à différents taux, entièrement compatible avec la norme iso11898.

3. Anti-interférence : le récepteur différentiel a une large gamme de résistance aux interférences des modes courants, fonction d'interférence électromagnétique (EMI).

4. Impédance d'entrée élevée : ce produit a une impédance d'entrée élevée, permettant 120 nœuds avec bus de protection anti-interférence instantanée.

5. Transmission : modes de veille à faible courant, le courant typique est de 370 μa. Le taux de transmission du signal est jusqu'à 1 mo/s.

Description :

Protection thermique, fonction de protection contre les défauts de circuit ouvert. Contrôle de la pente pour réduire les interférences de fréquence radio (RFI).[11]

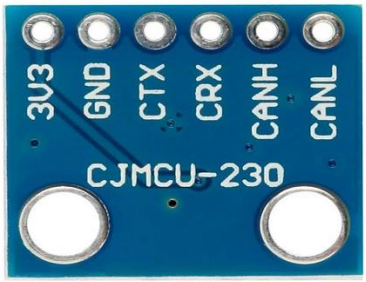

FIGURE 12: CAN TRANSCEIVER

III.5. Capteur de température MCP9808

Capteur de température basé sur le MCP9808 permettant de mesurer une température de -40 à +125 °C. Il communique avec un microcontrôleur type Arduino ou compatible via une liaison I2C.
Alimentation : 2,7 à 5,5 Vcc
Consommation : 200 µA
Plage de mesure : -40 à +125 °C
Précision : 0,5 °C
Interface I2C (adresse configurable) [12]

III.6. CAPTEUR DE LUMIERE/LUX I2C - VEML7700

Le capteur dispose d'une plage dynamique de 16 bits pour la détection de la lumière ambiante de 0 lux à environ 120 klux avec une résolution jusqu'à 0,0036 lx/ct, avec gain et temps d'intégration réglables par logiciel.

L'interfaçage est simple - ce capteur utilise l'I2C simple et universel. Nous avons placé ce capteur sur une carte d'alimentation avec un régulateur de 3,3 V et un sélecteur de niveau logique pour que vous puissiez l'utiliser avec des microcontrôleurs d'alimentation/logiques de 3,3 V ou 5 V. Nous avons écrit des bibliothèques pour Arduino (C/C++) ainsi que Circuit Python (Python 3) afin que vous puissiez utiliser ce capteur avec à peu près n'importe quel périphérique, même un Raspberry Pi ! [13]

III.7. MAX20340

Le MAX20340 est un circuit intégré de gestion de communication CPL bidirectionnel universel avec un débit de 166,7 kbps débit binaire maximal. L'appareil est capable d'un courant de charge maximum de 1,2 A.

Le MAX20340 dispose d'un circuit de détection d'esclave qui signale une interruption au système lorsque le maître PLC détecte la présence d'un automate esclave sur la ligne électrique. Cette fonction permet au système de rester dans un état de faible puissance jusqu'à ce qu'un esclave l'appareil est connecté. De nombreuses fonctionnalités du MAX20340, telles que le mode maître/esclave, l'adresse I2C, l'automate double/simple le mode esclave et l'adresse esclave de l'API sont configurables par broche. [14]

Configuration de l'appareil

Après la réinitialisation à la mise sous tension (POR), le mode maître/esclave, l'adresse esclave de l'automate (esclave uniquement), le mode d'adresse esclave de l'automate (maître uniquement) et l'adresse I2C sont configurées en fonction de la valeur de la résistance RSEL. L'état de la configuration peut être interrogé en lisant les bits I2C_ADD et PS_ADD du registre 0x05.

Conception et réalisation du projet

I. Architecture globale du projet

La conception de l'architecture globale du projet a été divisé sur trois grandes parties :

-L'acquisition des données des capteurs par STM32.

-La configuration et la mise en œuvre de la communication CAN entre les deux cartes embarqués STM32 et Raspberry pi.

-Le traitement des données acquises par la Raspberry et l'affichage sur la plateforme Domoticz.

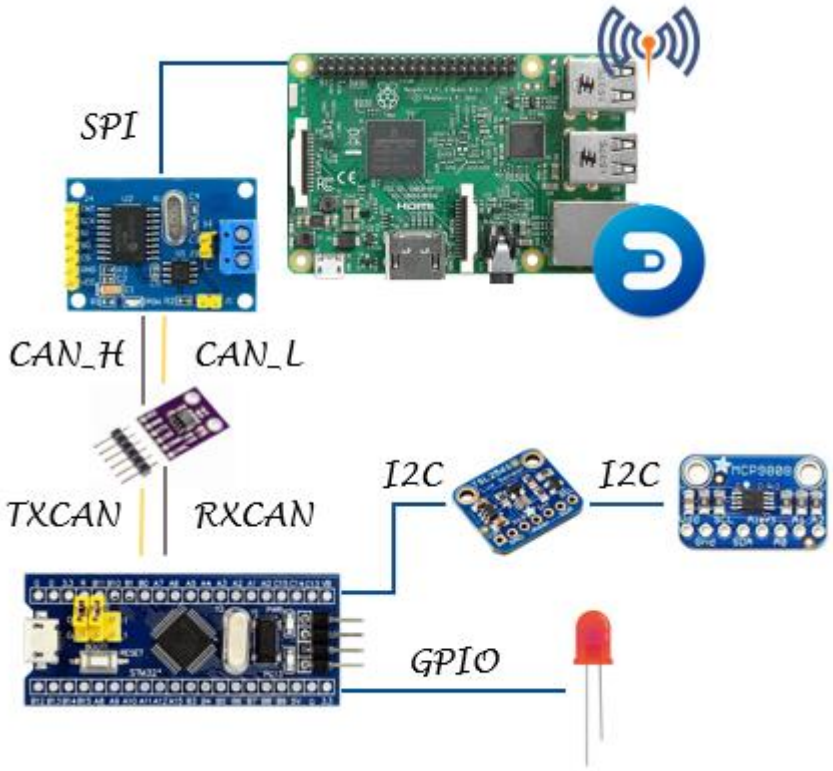

FIGURE 13: ARCHITECTURE GLOBALE

L'acquisition des données réalisée par les capteurs MCP9808 et VEML7700 se fait via le protocole I2C, alors la carte STM32 reçoit les données sous format i2c et les transforme en format TX RX CAN.

Le transceiver de sa part transforme les Tx et Rx CAN en CAN high et CAN low.

Le MCP contient à la base un transceiver qui reçoit la trame sous format CAN.

En effet, la Raspberry ne traite les trames CAN, donc pour cela on a utilisé le MCP2515 qui contient le SPI et qui va permettre à la Raspberry de récupérer la trame CAN via SPI.

Le traitement au sein de la Raspberry, permet de décortiquer la trame CAN et récupérer la donnée nette de la température ou/et la luminosité.

A l'aide du protocole MQTT, on utilise le principe (publish/subscribe) pour afficher les données sur la plateforme Home Assistant Domoticz.

Dans l'autre sens, pour contrôler les leds des salles qui sont liées à la carte STM, on envoie des requêtes MQTT (domoticz/out) qui seront reçues par le MCP et envoyés vers la carte STM via le transceiver.

Pour le protocole MQTT, on distingue trois topics utilisés :

- Domoticz/in
- Domoticz/out

II. Trames transmises

II.2. Trames I2C

Des capteurs vers la carte STM :

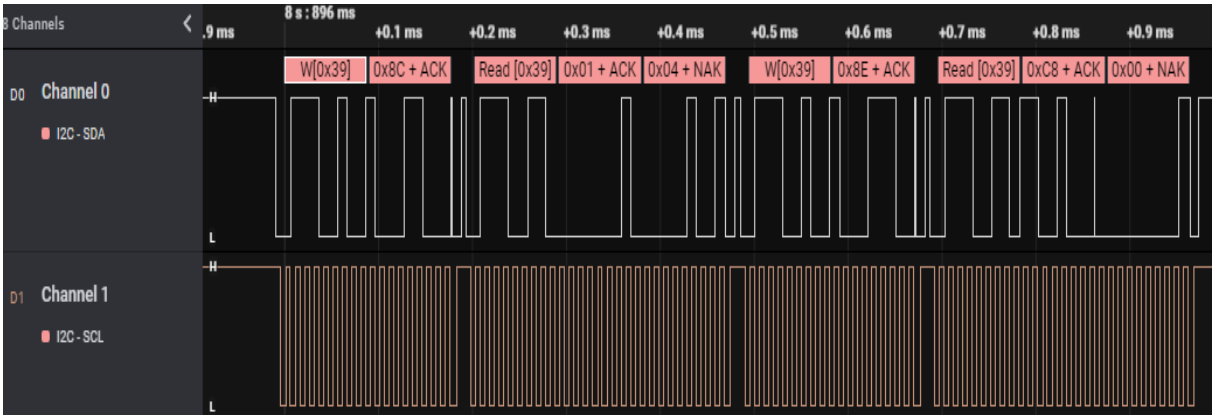

FIGURE 14: TRAME DE LUMINOSITÉ

Cette trame est ce que récupère la carte STM de la part du capteur de luminosité, pareil pour la trame de température.

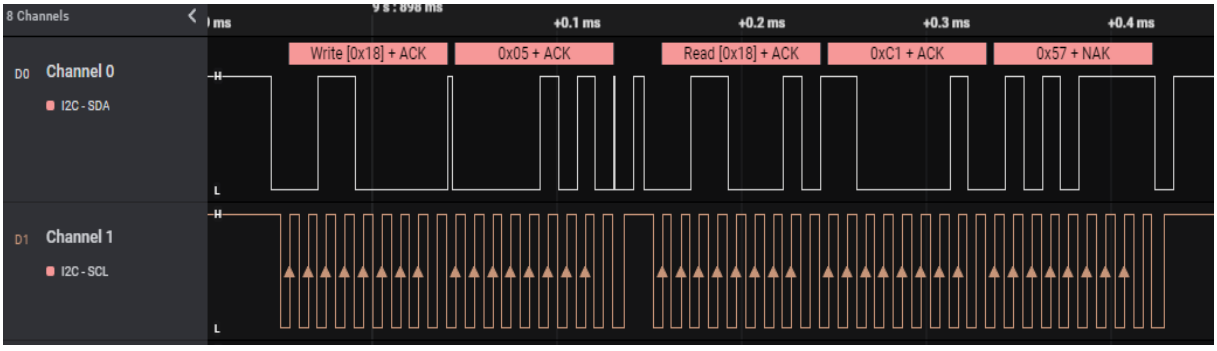

FIGURE 15: TRAME DE TEMPÉRATURE

II.2. Trames CAN

Les trames CAN qui se transmettent entre la carte STM et la Raspberry sont de la forme,

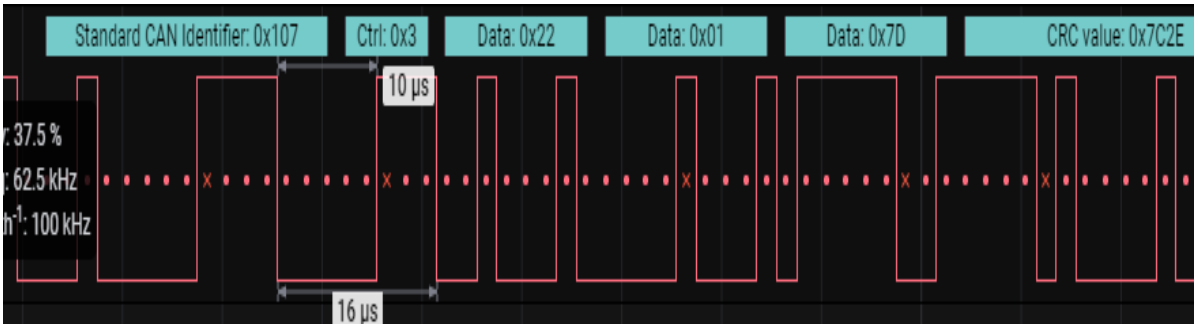

FIGURE 16: TRAME CAN DE LUMINOSITÉ

Luminosité:

-ID 0x22

-DATA: 4Bytes

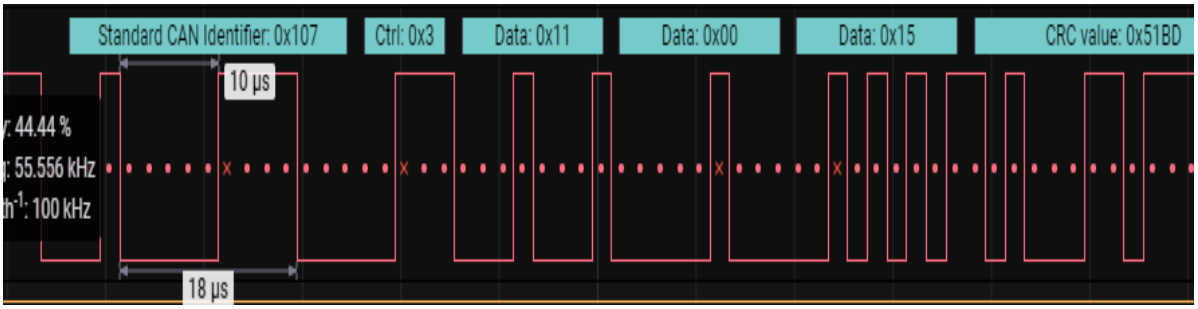

FIGURE 17: TRAME CAN DE TEMPERATURE

Temperature

-ID 0x11

-DATA: 4Bytes

III. Circuit réalisé

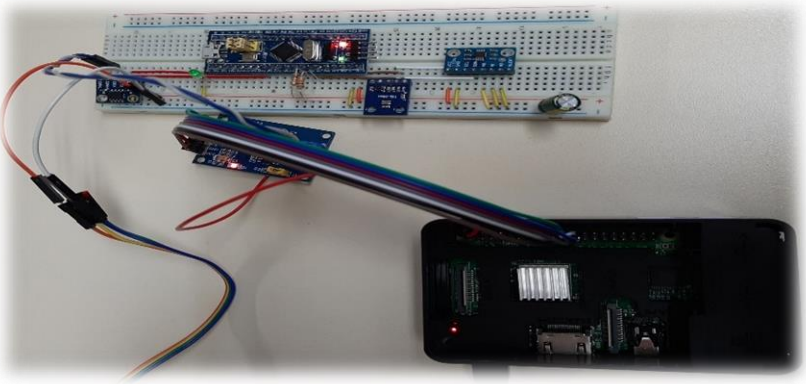

FIGURE 18: RÉALISATION MATÉRIELLE

Modélisation sur le logiciel fritzing

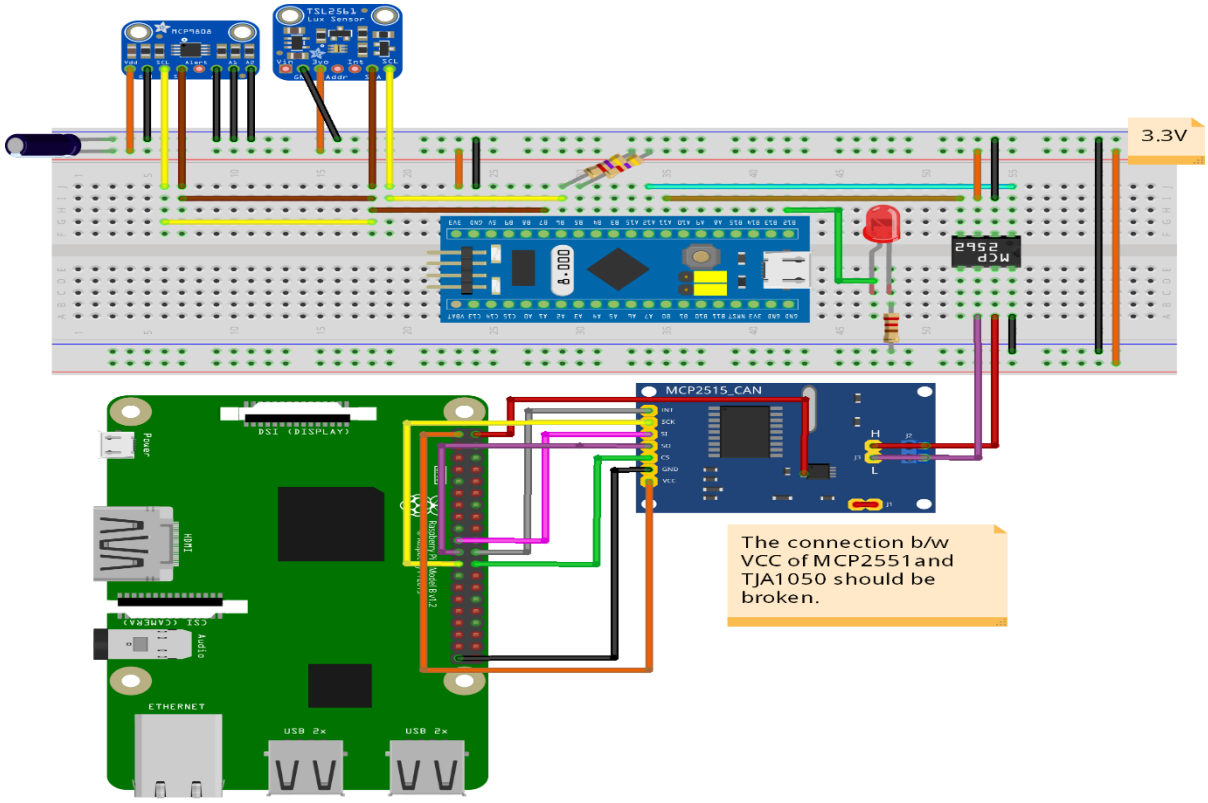

FIGURE 19: MODÉLISATION LOGICIELLE

IV. Résultats

Les données acquises de la part des capteurs ont comme valeurs température à titre d'exemple à un instant t 29°C et de luminosité 222 lux, en exécutant le code, elles vont être affichées dans la plateforme Domoticz.

```
pi@raspberrypi:~/Projet $ python3 code.py
CAN Rx test
Bring up CAN0....
Ready
MQTT: Connected to server
New Luminosity Data:217
New Temp Data:30
New Luminosity Data:217
New Temp Data:30
New Luminosity Data:217
New Temp Data:30
New Luminosity Data:218
New Temp Data:29
```

FIGURE 20: ACQUISITION DES DONNÉES

Résultats affichés sur Domoticz :

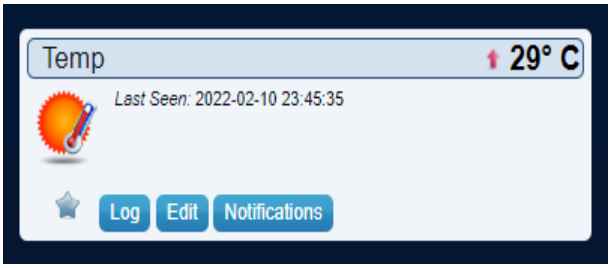

FIGURE 21 : TEMPÉRATURE

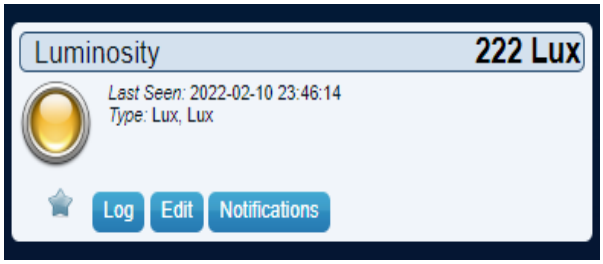

FIGURE 22 : LUMINOSITÉ

Pour le contrôle de la led, on l'allume sur Domoticz comme suit :

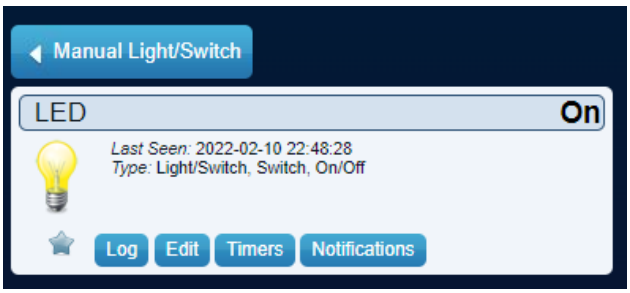

FIGURE 21: LED CONTRÔLE

Conclusion

En guise de conclusion, on a pu réaliser trois tâches parmi quatre, les trois premières sont détaillées dans le rapport, la quatrième partie qu'on n'a pas pu réaliser concerne la communication « single wire can » à cause de plusieurs contraintes, premièrement le MCP contient le transceiver et un MCP ce qui nous empêche d'agir sur le CAN_L et CAN_H, et deuxièmement nous ne possédons pas un transceiver single wire qui est un matériel principal pour réussir cette tâche. Cette tache peut être l'un des perspectifs de ce projet dans l'avenir.

WEBOGRAPHIE

[1] https://www.cesbio.cnrs.fr/lancement-du-groupement-dinteret-scientifique-neocampus/

[2] https://www.irit.fr/~Zoubir.Mammeri/Cours/Introduction_CAN.pdf

[3] http://projet.eu.org/pedago/sin/term/8-bus_SPI.pdf

[4] https://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-I2C.htm

[5] https://www.irit.fr/~Philippe.Truillet/zilab/madagascar/supports/MQTT-1.2.pdf

[6] https://domoticz.com/

[7] https://www.journaldunet.fr/web-tech/dictionnaire-de-l-iot/1440686-mqtt-comment-fonctionne-ce-protocole/#:~:text=Mosquitto%20est%20le%20broker%20le,XP%2D10%2C%20Linux).

[8] https://letmeknow.fr/fr/cartes-programmables/2042-carte-de-developpement-stm32f103c8t6-arm-stm32-bluepill.html

[9] https://fr.wikipedia.org/wiki/Raspberry_Pi#/media/Fichier:Raspberry_Pi_-_Model_A.jpg

[10] https://letmeknow.fr/fr/autres-composants/2134-module-can-bus-mcp2515.html

[11] https://www.infineon.com/cms/en/product/transceivers/

[12]https://fr.aliexpress.com/item/1005003640723098.html?

[13]https://boutique.semageek.com/fr/1550-capteur-de-lumierelux-i2c-veml7700-3008172073133.html

[14] https://datasheets.maximintegrated.com/en/ds/MAX20340.pdf