



ECOLE NATIONAL
D'INGENIEURSDE SFAX



République Tunisienne
Ministère de l'enseignement Supérieur de la recherche scientifique et de
technologie

PROJET DE FIN D'ANNEE

Intitulé :

ETUDE D'UNE CONVERSION CAN/ETHERNET

Réalisé PAR :

ELOUNI LOUAI

SAOUDI OUMAYMA

ELLOUMI SALAH ALAEDDINE

DABBECH ZAINEB

SOUTENU PUBLIQUEMENT LE 02 JUIN 2023, DEVANT LE JURY COMPOSE DE :

EXAMINATEUR : M.C.SALLEM SOUHIR

ENCADRANTE : PR. TRABELSI HAFETDH

Année universitaire

2022/2023

Remerciements

Nous tenons à exprimée tout d'abord nos sincères remerciement à notre encadreur Pr. **TRABELSI HAFEDH** d'avoir accepté de diriger notre projet de fin d'année aussi pour la qualité de son encadrement, ses conseils intéressants et ses encouragement.

Nous remercions, également M.C. **SALLEM SOUHIR** de nous faire l'honneur de présider le jury de notre PFA. Il nous parait tout naturel d'adresser nos vifs remerciements à toutes les personnes qui ont apporté leur soutien.

Sommaire

I.	INTRODUCTION GENERALE :	1
II.	Principe d'une conversion CAN/Ethernet proposée.....	2
II.1.	Description :	2
II.2.	Schéma de principe de la conversion CAN/ETHERNET.....	2
III.	Aspect matériel :	3
III.1.	La carte Arduino nano :	3
III.2.	La carte stm32 :	3
III.3.	Le module W5100:	4
III.4.	La carte Rasbery pi 2 :	4
III.5.	Module MCP2515:	6
III.6.	Module MCP2551:	6
III.7.	Potentiomètre:	6
III.8.	Câble RJ-45:	6
IV.	Les Protocoles CAN et Ethernet :	7
IV.1.	Protocole CAN :	7
IV.2.	Protocole Ethernet :	8
V.	Présentation du système de conversion CAN/Ethernet :	10
V.1.	Schème de câblage :	10
V.2.	Avantages et limites :	13
V.1.1.	Les avantage :	13
V.1.2.	Les limite :	14
VI.	Partie programmation:	15
VI.1.	Acquisition des données et transmission à travers bus CAN	15
VI.2.	Réception en bus CAN et conversion et transmission en Ethernet :	16
VI.3.	Affichage des données :	17
VII.	Conclusion générale :	18
ANNEXE1	19
ANNEXE2	20
ANNEXE3	21
ANNEXE4	23
BIBLIOGRAPHIE	24

Liste des figures

Figure 1:Schéma fonctionnel.....	2
Figure 3: Pinout de la carte Arduino	3
Figure 2:Carte Arduino nano.....	3
Figure 4:carte STM32F103C8.....	4
Figure 6: Carte raspberry pi2.....	4
Figure 5:Module W5100	4
Figure 7: Pin_out de la carte raspberry	5
Figure 8: Module MCP2515.....	6
Figure 9:Module MCP2551.....	6
Figure 10:Potentiomètre	6
Figure 11: Câble RJ_45.....	6
Figure 12:Trame CAN.....	7
Figure 13: Trame Ethernet	8
Figure 14 : Description du système de conversion CAN /Ethernet.....	10
Figure 15: Développement par le logiciel fritzing	12
Figure 16: Trame CAN.....	15

I. INTRODUCTION GENERALE :

L'intégration de la technologie Ethernet dans l'industrie a transformé la manière dont les entreprises gèrent leurs systèmes et leurs processus de production. Dans un tel contexte, Ethernet joue un rôle essentiel en permettant la connectivité et la communication fiables entre les équipements industriels et au niveau des réseaux automobiles.

Le protocole CAN (Controller Area Network) et Ethernet sont deux technologies clés utilisées dans l'industrie automobile pour les réseaux embarqués. Le protocole CAN est largement utilisé depuis plusieurs décennies et constitue le fondement des systèmes de communication dans les véhicules. Il permet une transmission de données à haut débit tout en garantissant une latence minimale, ce qui est essentiel pour des systèmes temps réel critiques. D'autre part, Ethernet gagne également en popularité dans l'industrie automobile en tant que technologie de réseau embarqué. Les deux protocoles jouent des rôles complémentaires dans les réseaux embarqués. Avec l'évolution de l'industrie automobile vers des véhicules de plus en plus connectés et autonomes, une combinaison efficace de ces deux technologies devient essentielle pour garantir des performances optimales et une sécurité accrue sur la route.

Dans cet esprit, on s'intéresse à faire une étude de conversion d'une trame CAN vers une trame Ethernet et de visualiser des données transmises (tension aux bornes du potentiomètre) sur un écran.

Ce présent rapport de Projet de Fin d'Année PFA comporte trois parties.

- Dans une première partie, on présente une idée globale sur le projet en générale.
- Dans une deuxième partie, on s'intéresse à une étude bibliographique qui porte sur l'aspect matériel.
- Dans une troisième partie, on présente les codes de programme développés

Ce travail sera clôturé par une conclusion générale.

II. Principe d'une conversion CAN/Ethernet proposée

II.1. Description :

Afin d'étudier les protocoles de communication CAN (Controller Area Network) et Ethernet, nous avons choisi dans le cadre de notre projet de PFA de faire la Conception et le développement d'un prototype de conversion CAN/Ethernet.

Ce prototype consiste à programmer une carte STM32F103C8T6 pour faire la conversion d'une trame CAN vers une trame Ethernet. Pour tester ce programme il nous a fallu développer deux circuits additionnels:

- Un circuit de génération des trames CAN pour simuler la trame d'entrée CAN qui va être reçue par la carte STM32. Ce circuit est composé d'un potentiomètre, une carte Arduino Nano, un module MCP2515 (Stand-Alone CAN Controller with SPI Interface)+TJA1050, un Module MCP2551 (CAN transceiver).

- Un circuit de réception de la trame Ethernet transmise par la carte STM32 et L'affichage des données transmises sur un écran. Ce circuit est composé d'un

Module Shield Ethernet pour Arduino W5100 (interface SPI/Ethernet) et d'une carte Raspberry Pi 2.

II.2. Schéma de principe de la conversion CAN/ETHERNET

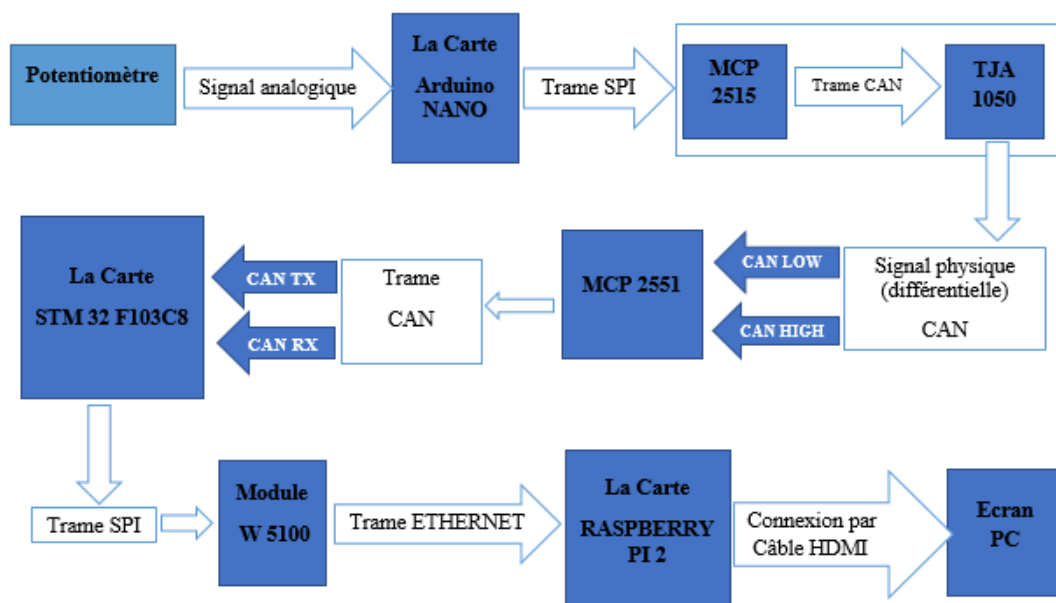


Figure 1:Schéma fonctionnel

III. Aspect matériel :

III.1. La carte Arduino nano :

La carte Arduino Nano est une petite carte de développement basée sur le microcontrôleur Atmel AVR. Elle est similaire à la carte Arduino Uno, mais dans un format plus compact. Il convient de noter qu'il existe différents modèles et versions de la carte Arduino Nano, donc les caractéristiques spécifiques peuvent varier légèrement en fonction de la version choisie. (Voir ANNEXE 1)

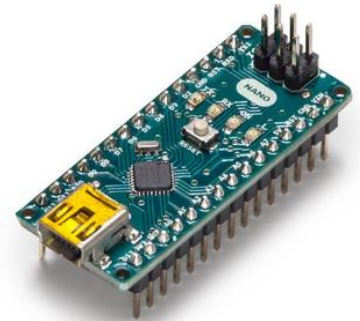


Figure 2: Carte Arduino nano

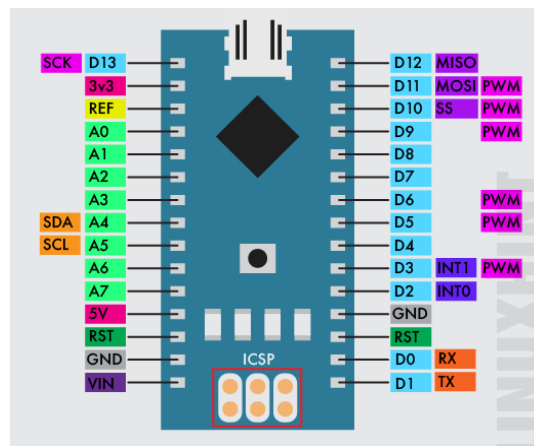


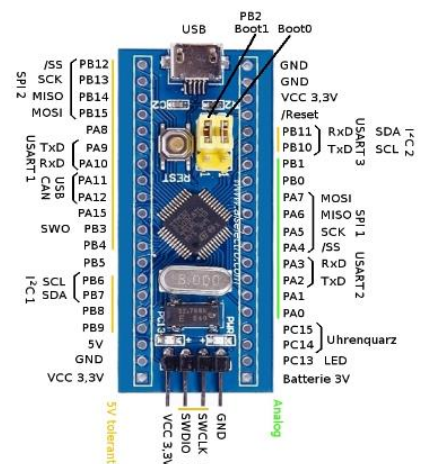
Figure 3: Pinout de la carte Arduino

III.2. La carte stm32 :

C'est une carte de développement basée sur la famille de microcontrôleurs STM32 de STMicroelectronics. Ces cartes sont conçues pour aider les développeurs à créer des prototypes et des applications embarquées rapidement et facilement.

La carte STM32F103C8 est une carte de développement basée sur le microcontrôleur STM32F103C8T6, qui fait partie de la famille STM32F1 de STMicroelectronics.

(Voir ANNEXE 1)



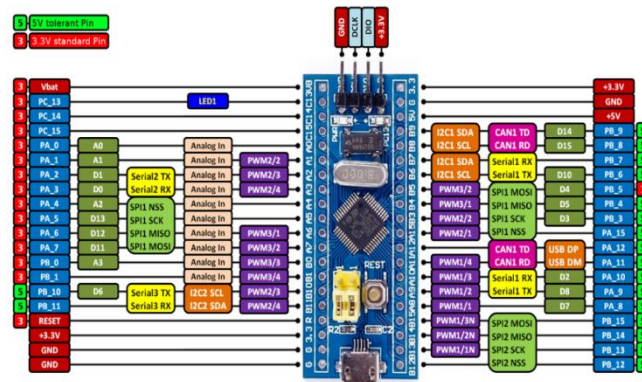


Figure 4:carte STM32F103C8

III.3. Le module W5100:

Le W5100 est un module réseau qui permet d'ajouter facilement une connectivité Ethernet aux systèmes basés sur des microcontrôleurs tels qu'Arduino, Raspberry Pi et d'autres plates-formes similaires. Il s'agit d'une pile matérielle complète de protocoles TCP/IP qui prend en charge plusieurs connexions et offre une interface de programmation simple basée sur des sockets.



Figure 5:Module W5100

III.4. La carte Rasbery pi 2 :

La carte Raspberry Pi 2 (Voir ANNEXE 1) est une ordinateuse mono carte de petite taille, développé par la fondation Raspberry Pi. C'est une carte polyvalente qui peut être utilisé pour de nombreuses applications différentes.



Figure 6: Carte rasberry pi2

RO: Il existe plusieurs différences entre les différentes cartes Raspberry Pi, qui se distinguent notamment par leurs spécifications techniques, leurs capacités de traitement, leurs capacités de stockage, leurs ports d'entrée/sortie et leurs fonctionnalités. Mais si on est besoin de la carte la moins chère possible, la Raspberry Pi 2 peut être une option plus économique.

Dans notre application on va utiliser :

- **Langage de programmation :** " PYTHON " car c'est le plus couramment utilisés avec la Raspberry Pi 2, facile à apprendre et est souvent utilisé pour créer des programmes destinés à la robotique, l'IoT et l'analyse de données.
- **Systeme d'exploitation :** " RASPBIAN " c'est une distribution Linux basée sur Debian et spécialement conçue pour la Raspberry Pi. Il est facile à utiliser et dispose d'une grande communauté de développeurs. Voici quelques-unes des fonctionnalités et caractéristiques clés de la Raspberry Pi 2 :

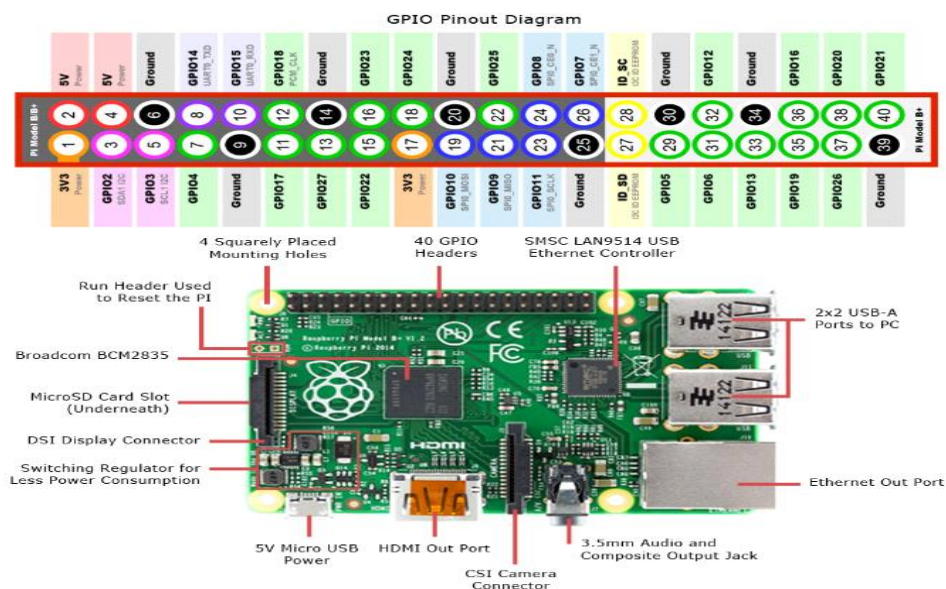


Figure 7: Pin_out de la carte raspberry

III.5. Module MCP2515:

Le module MCP2515 intègre le CI MCP2515 et fournit des broches d'entrée/sortie ainsi qu'une interface de communication, généralement via le protocole SPI (Serial Peripheral Interface). Il permet de connecter facilement le MCP2515 à un microcontrôleur ou à d'autres dispositifs. ce module permet de bénéficier des fonctionnalités du MCP2515, telles que la prise en charge du protocole CAN 2.0B, la transmission et la réception de messages CAN



Figure 8: Module MCP2515

III.6. Module MCP2551:

Le module MCP2551 est une version du MCP2551 intégré dans un format de module prêt à l'emploi. Il facilite l'intégration du MCP2551 dans un projet, car il fournit une carte de circuit imprimé avec le MCP2551 déjà monté et des connecteurs pour les broches d'entrée/sortie

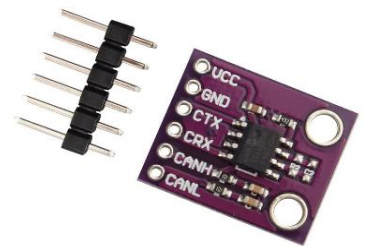


Figure 9:Module MCP2551

III.7. Potentiomètre:

Un potentiomètre est une résistance variable. Les potentiomètres permettent de réguler et modifier le flux de courant circulant dans un circuit.



Figure 10:Potentiomètre

III.8. Câble RJ-45:

Le câble RJ45 est un câble destiné à relier des ordinateurs sur un réseau chez soi ou en entreprise.



Figure 11: Câble RJ_45

IV. Les Protocoles CAN et Ethernet :

IV.1. Protocole CAN :

Le CAN utilise une topologie de réseau en bus, où les nœuds sont connectés à une ligne de communication commune. Les nœuds peuvent émettre des signaux électriques dominants ou récessifs sur le bus, et en cas de collision, le signal dominant prévaut.

Chaque message transmis sur le bus CAN est identifié par un identifiant unique. Les messages sont priorisés en fonction de leurs identifiants, et en cas de collision, le message avec l'identifiant le plus bas est transmis en premier.

Les données sont transmises sous forme de trames CAN. Il existe deux formats de trame : le format standard (CAN 2.0A) et le format étendu (CAN 2.0B). Les trames contiennent des bits de contrôle, des bits de données et des bits de CRC pour la détection d'erreurs.

⇒ Dans notre cas on va utiliser le format étendu CAN 2.0B.

Le protocole CAN intègre des mécanismes de détection et de correction d'erreurs pour assurer une transmission fiable. Chaque trame est vérifiée par le destinataire à l'aide d'un code de redondance cyclique (CRC).

Le CAN offre différentes vitesses de transmission, exprimées en bauds. Les vitesses typiques sont de 125 kbit/s, 250 kbit/s, 500 kbit/s et 1 Mbit/s. La sélection de la vitesse appropriée dépend des exigences du système.

→ Dans notre cas on va utiliser la vitesse 500Kbits/s.

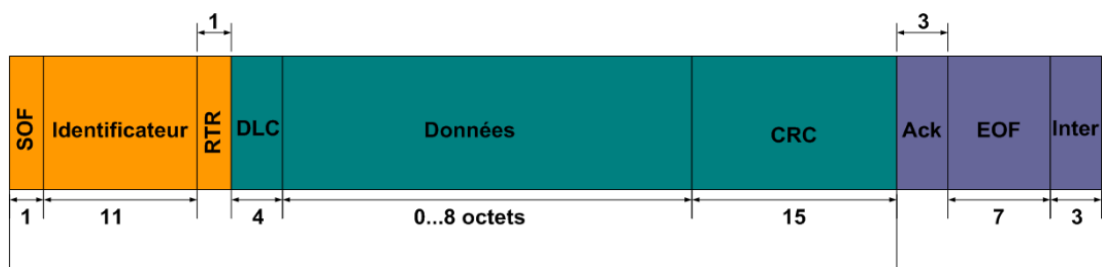


Figure 12:Trame CAN

IV.2. Protocole Ethernet :

Le protocole Ethernet est un ensemble de règles et de normes utilisées pour la transmission de données sur un réseau local (LAN). Il définit les aspects physiques, les spécifications du câblage, les protocoles de couche liaison de données et les mécanismes de contrôle d'accès au support (MAC) utilisés pour la communication. Ethernet est l'une des technologies de réseau les plus couramment utilisées et offre une méthode standardisée et fiable pour l'acheminement des données entre les périphériques connectés à un réseau Ethernet. Il repose sur une architecture en couches, où chaque couche est responsable d'une fonction spécifique dans le processus de transmission des données.

Le protocole Ethernet utilise des trames Ethernet, qui sont des structures binaires contenant des informations telles que les adresses de destination et source, le type de protocole utilisé et les données à transmettre. Les adresses MAC sont utilisées pour l'acheminement des trames vers la destination appropriée. Les commutateurs Ethernet sont des dispositifs réseau importants dans le protocole Ethernet. Ils permettent la transmission des trames Ethernet vers les périphériques de destination appropriés en fonction de leurs adresses MAC.

Ethernet offre différents débits de transmission, allant de 10 Mbps (Ethernet) à des débits plus élevés tels que 10 Gbps (10 Gigabit Ethernet) et 100 Gbps (100 Gigabit Ethernet). Ces débits permettent des communications rapides et efficaces sur les réseaux locaux.

Voici un exemple de trame Ethernet et une explication de chaque partie :

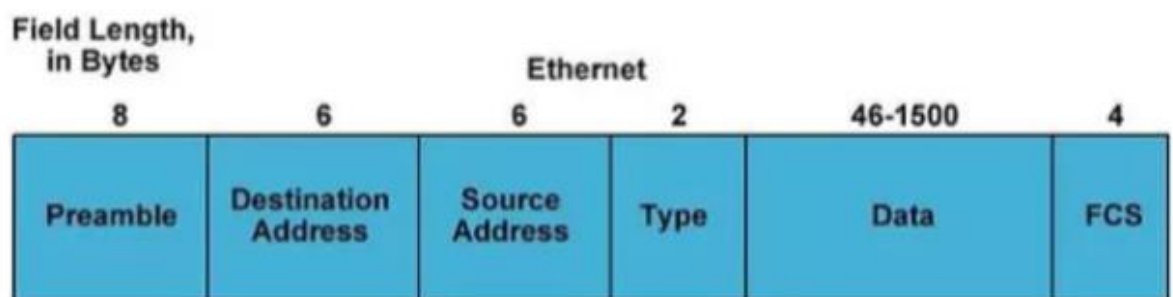


Figure 13: Trame Ethernet

1. **Le préambule** est composé de 8 octets (64 bits) de valeurs alternées (10101010 et 01010101), ce qui permet aux périphériques de se synchroniser avant de commencer la transmission de données.

2. **Adresse de destination** (Destination Address) : Ce champ contient l'adresse MAC (Media Access Control) du périphérique destinataire. Il est composé de 6 octets (48 bits) et permet au commutateur Ethernet de diriger la trame vers le périphérique approprié.

3. **Adresse source** (Source Address) : Ce champ contient l'adresse MAC du périphérique émetteur de la trame. Il est également composé de 6 octets (48 bits) et permet au destinataire de savoir d'où provient la trame.

4. **Type** : Ce champ de 2 octets indique le type de protocole utilisé dans les données encapsulées dans la trame. Par exemple, il peut indiquer si les données sont du protocole IP (Internet Protocol) ou d'un autre protocole réseau.

5. **Données** (Data) : Ce champ contient les données réelles à transmettre, qui peuvent être de différentes longueurs, allant de 46 à 1500 octets. Les données peuvent inclure des informations telles que des paquets IP, des segments TCP, des datagrammes UDP, etc., selon le protocole spécifié dans le champ Type.

6. **FCS** (Frame Check Sequence) : Il s'agit d'un champ de 4 octets utilisé pour détecter les erreurs de transmission dans la trame Ethernet. Le FCS est calculé à partir des autres parties de la trame et permet au destinataire de vérifier l'intégrité des données reçues.

V. Présentation du système de conversion CAN/Ethernet :

V.1. Schème de câblage :

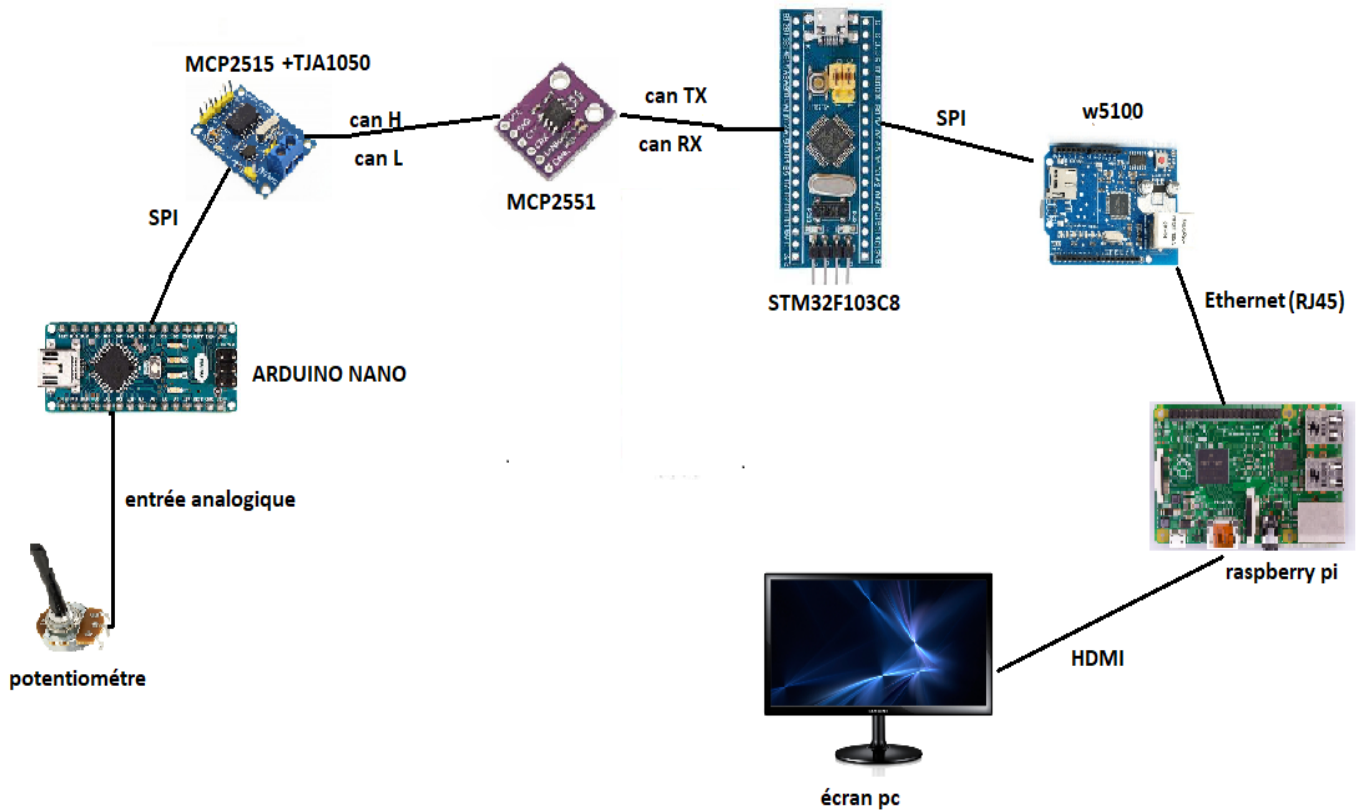


Figure 14 : Description du système de conversion CAN /Ethernet

La figure 14 présente un potentiomètre connecté à une entrée analogique de la carte Arduino Nano pour convertir la tension aux bornes de potentiomètre en valeur numérique qui sera envoyé vers le module MCP2515 par le protocole Serial Peripheral Interface SPI à l'aide d'un code écrit en langages de programmation C/C++ sur l'IDE Arduino IDE.

Le module MCP2515+TJA1050 convertit à son tour la trame SPI en une trame CAN. En effet, les lignes CANH H(CAN High) et CAN L(CAN Low) du MCP2515 sont connectés aux lignes CAN H et CAN L du module MCP2551(High Speed CAN).

On connecte le CAN TX et CAN RX du MCP2551 aux pins CAN RX CAN TX (PB8 et PB9) de la carte STM32F407VG.

Un code en langage de programmation C implémenté dans la carte STM32 qui prélève la valeur de la tension aux bornes du potentiomètre de la trame CAN reçue et la met dans le champ des données de la trame Ethernet. On connecte les pins de communication SPI de la carte STM32F103C8 aux pins du module SPI W5100 (puisque on n'a pas de port Ethernet physique sur cette carte STM32 qui supporte un câble Ethernet RJ-45).

Ce dernier W5100 par le biais du port Ethernet (RJ-45) est branché au port Ethernet d'une RASPBERRY PI 2.

Un code en langage Python sur la RASPBERRY PI 2 pour afficher la valeur de la tension du potentiomètre sur l'écran.

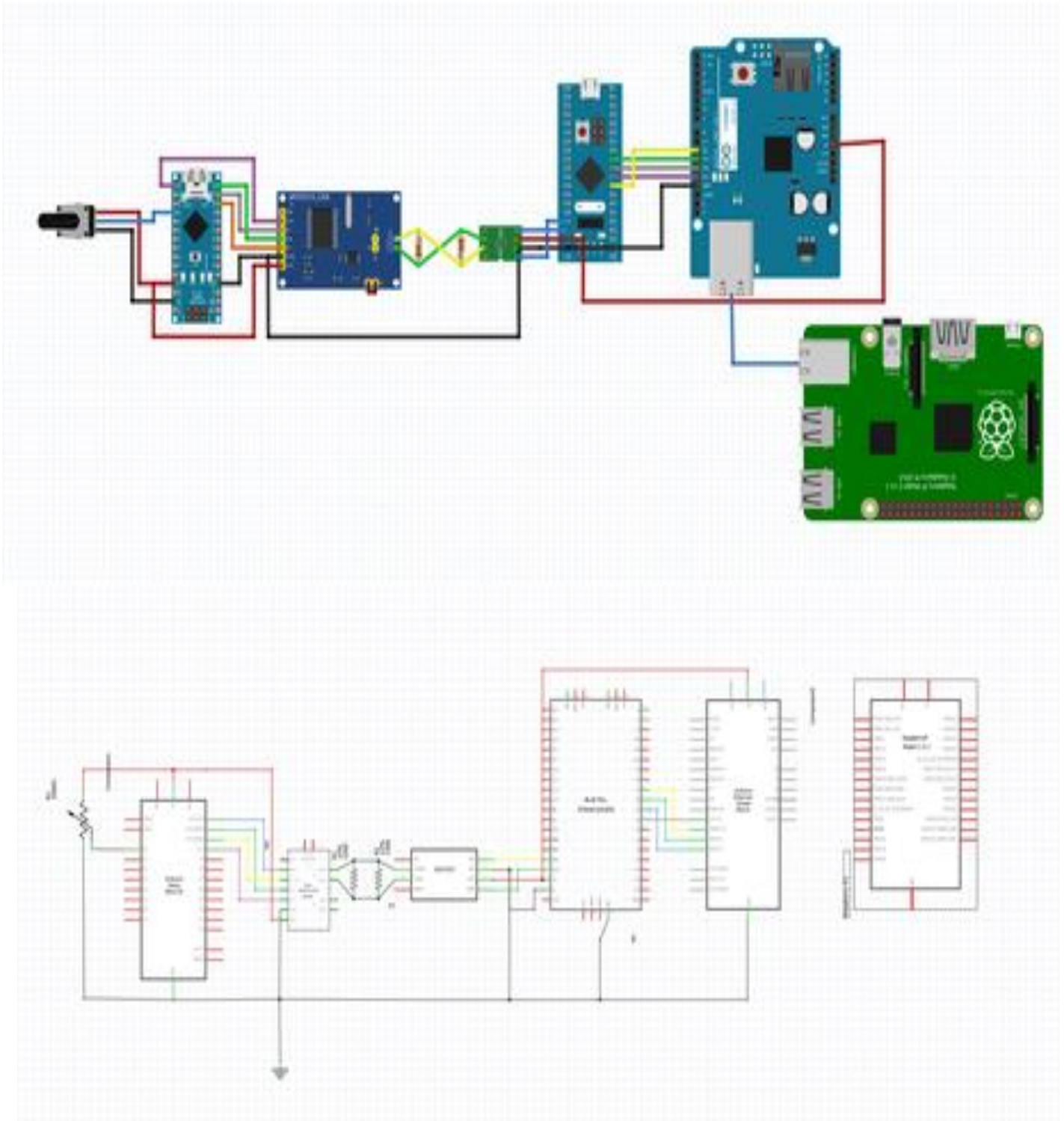




Figure 15: Développement par le logiciel fritzing


V.2. Avantages et limites :


Le prototype de conversion CAN/ETHERNET permet de relier un réseau CAN (Controller Area Network) à un réseau Ethernet, ce qui offre certains avantages mais présente également des limites. Voici un aperçu de ces avantages et limites :

V.1.1. Les avantage :


 **Connectivité étendue** : La conversion CAN/ETHERNET permet d'étendre la connectivité du réseau CAN en le reliant à un réseau Ethernet. Cela facilite l'intégration de systèmes CAN avec d'autres équipements ou systèmes informatiques utilisant Ethernet, tels que le domaine automobile.


 **Transmission à longue distance** : Ethernet offre une meilleure capacité de transmission sur de longues distances par rapport au CAN. En convertissant le CAN en Ethernet, il devient possible de transmettre les données CAN sur de plus grandes distances, ce qui est particulièrement utile lorsque les nœuds CAN sont dispersés sur un vaste emplacement géographique.


 **Interopérabilité** : Ethernet est un protocole standard largement utilisé, ce qui signifie que les dispositifs compatibles Ethernet peuvent facilement communiquer avec les données CAN converties. Cela permet une intégration plus aisée avec d'autres systèmes et dispositifs prenant en charge Ethernet.


 **Infrastructure existante** : La conversion CAN/ETHERNET peut profiter de l'infrastructure Ethernet déjà en place dans de nombreux environnements. Les réseaux Ethernet sont couramment utilisés dans les entreprises, les usines et les systèmes informatiques, ce qui signifie que l'installation de nouveaux câbles ou équipements spécifiques pour le CAN peut souvent être évitée.

V.1.2. Les limite :

 **Latence accrue** : La conversion du CAN en Ethernet peut introduire une latence supplémentaire par rapport à une communication directe en CAN. Les protocoles de conversion et les équipements intermédiaires peuvent ajouter une certaine surcharge qui peut entraîner un délai plus long dans la transmission des données CAN.

 **Complexité accrue** : La mise en place d'une conversion CAN/ETHERNET nécessite des équipements supplémentaires tels que des convertisseurs, des passerelles ou des modules de communication spécifiques. Cela ajoute de la complexité au système global et peut nécessiter une configuration et une maintenance supplémentaires.

 **Coûts** : Les équipements nécessaires pour la conversion CAN/ETHERNET peuvent entraîner des coûts supplémentaires par rapport à une communication CAN autonome. Les convertisseurs ou passerelles peuvent avoir un prix plus élevé, en particulier si des fonctionnalités avancées sont nécessaires.

 **Fiabilité** : L'introduction de la conversion CAN/ETHERNET peut potentiellement augmenter les points de défaillance du système. Des problèmes liés à la conversion ou aux équipements Ethernet peuvent avoir un impact sur la fiabilité globale de la communication CAN.

➔ Il est important de noter que les avantages et les limites de la conversion CAN/ETHERNET peuvent varier en fonction des spécificités de chaque application et de la qualité des équipements utilisés.

VI. Partie programmation:

VI.1. Acquisition des données et transmission à travers bus CAN

Ce code (Voir ANNEXE2) est écrit en langage C++. Dans ce code on utilise les bibliothèques SPI et MCP2515 pour la communication SPI et CAN.

Le code est structuré comme suit :

- ❖ Les directives **#include** incluent les bibliothèques nécessaires pour utiliser SPI (**SPI.h**) et la communication CAN (**mcp2515.h**).
- ❖ La ligne **#define potPin A0** définit le numéro de broche analogique à utiliser pour la lecture d'une valeur de potentiomètre.
- ❖ La variable **potValue** est déclarée pour stocker la valeur du potentiomètre.
- ❖ La structure **can_frame** est définie pour représenter un message CAN.
- ❖ L'objet **mcp2515** de la classe MCP2515 est créé en spécifiant le numéro de broche de sélection du circuit intégré (CS pin) qui est 10.
- ❖ La fonction **setup ()** est exécutée une seule fois au démarrage de l'Arduino. Elle initialise la communication série (**Serial.begin()**) à une vitesse de 9600 bauds, démarre la communication SPI (**SPI.begin()**), réinitialise le MCP2515 (**mcp2515.reset()**), configure la vitesse de communication CAN (**mcp2515.setBitrate()**) à 500 Kbit/s avec une horloge de 8 MHz, et passe en mode normal (**mcp2515.setNormalMode()**).
- ❖ La structure **canMsg** est initialisée avec un identifiant CAN de **0x036**, une longueur de données de 8 octets, et les octets de données sont remplis avec des valeurs spécifiques : 1^{er} octet contient la valeur du potentiomètre et les autres octets sont à 0.

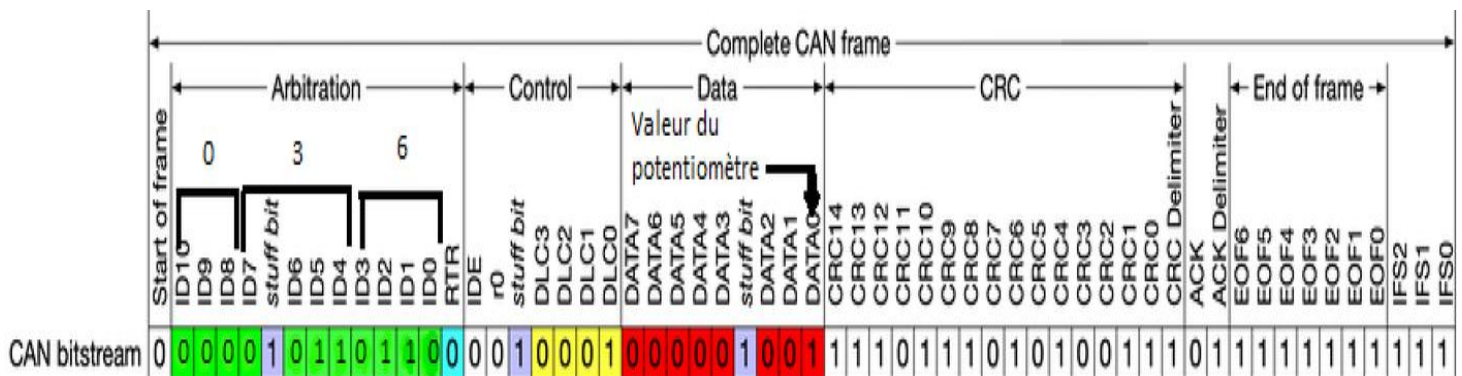


Figure 16: Trame CAN

❖ La fonction **loop()** est exécutée en boucle. Elle lit la valeur du potentiomètre avec **analogRead(potPin)**, effectue une mise à l'échelle de cette valeur à une plage de **0 à 255** avec **map()**, puis l'affiche via la communication série. Ensuite, elle envoie le message CAN avec la valeur du potentiomètre mise à jour en utilisant **mcp2515.sendMessage(&canMsg)**. Après une pause de 200 millisecondes avec **delay(200)**, le processus recommence.

⇒ Ce code permet de lire la valeur d'un potentiomètre, de l'afficher sur le moniteur série (écran du pc) et de l'envoyer via le SPI vers le module MCP2515+TJA1050 qui fournit l'information en signaux bus CAN.

VI.2. Réception en bus CAN et conversion et transmission en Ethernet :

Ce code (voir ANNEXE 3) implémenté sur une carte STM32F103C8 permet de transmettre une trame SPI après conversion de la trame CAN.

La carte STM32 envoie cette trame vers le module W5100 qui fait une autre conversion SPI/Ethernet.

Le programme est structuré comme suit :

Les directives « include main.h » incluent les fichiers d'en-tête nécessaires pour utiliser les bibliothèques STM32F10x et les fonctions Ethernet.

Les macros **MAC_SOURCE_ADDR** et **MAC_DEST_ADDR** définissent les adresses MAC source et destination de la trame Ethernet.

Les macros **IP_SOURCE_ADDR** et **IP_DEST_ADDR** définissent les adresses IP source et destination de la trame Ethernet.

La fonction **ETH_Configuration** est utilisée pour initialiser le contrôleur Ethernet avec les paramètres appropriés. Elle configure des paramètres tels que la négociation automatique de la vitesse, le mode duplex, la réception des trames de diffusion, etc. Elle configure également les adresses MAC et IP.

La fonction **ETH_SendFrame** est utilisée pour envoyer une trame Ethernet. Elle prend en paramètre un pointeur vers les données de la trame et sa longueur. Cette fonction utilise les descripteurs DMA (Direct Memory Access) pour transmettre les données via le contrôleur Ethernet.

VI.3. Affichage des données :

Ce code (Voir ANNEXE4) Python pour l'affichage des données reçues via Ethernet sur le terminal du Raspberry Pi 2, avec indication du système d'exploitation utilisé :

- ❖ Ce code utilise la bibliothèque socket pour la communication réseau via Ethernet entre le Raspberry Pi 2 et le périphérique distant.

- ❖ Le code lit en continu les données reçues sur le port défini et les affiche sur le terminal du Raspberry Pi 2 en utilisant la fonction print().

- ❖ La variable platform.system() est utilisée pour afficher le nom du système d'exploitation utilisé.

➔ Les données reçues doivent être en format texte (par exemple, des chaînes de caractères) pour pouvoir être affichées correctement.

➔ Vous pouvez modifier l'adresse IP et le port utilisés en fonction de votre configuration réseau.

VII. Conclusion générale :

Le développement de ce PFA nous a permis de connaître de près une application industrielle pertinente qui consiste à convertir une trame CAN en une trame Ethernet.

En premier lieu nous avons présenté les protocoles CAN et Ethernet et précisé les trames de chacun.

En deuxième lieu, nous avons développé un système matériel qui permet de réaliser cette conversion.

En troisième lieu, nous avons élaboré les différents codes pour les différentes cartes (Arduino , STM32 , Raspberry pi2) en utilisant les bibliothèques nécessaires pour intégrer les module MCP2515+TJA1050 , MCP2551et le W5100.

Comme perspectives, nous allons multiplier le nombre de nœuds.

ANNEXE1

Specifications:

Microcontroller	Atmel ATmega168 or ATmega328
Operating Voltage (logic level)	5 V
Input Voltage (recommended)	7-12 V
Input Voltage (limits)	6-20 V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	8
DC Current per I/O Pin	40 mA
Flash Memory	16 KB (ATmega168) or 32 KB (ATmega328) of which 2 KB used by bootloader
SRAM	1 KB (ATmega168) or 2 KB (ATmega328)
EEPROM	512 bytes (ATmega168) or 1 KB (ATmega328)
Clock Speed	16 MHz
Dimensions	0.73" x 1.70"

Raspberry Pi 2

35€

Ram : 1 Go

Nombre de processeur : 4

Processeur : ARMv7 (~6x plus puissant)

Cadence du processeur : 900 Mhz

Supporte Windows 10 : Oui

Stockage : Carte MicroSD

Ports : 4 USB 2.0

Puissance : 600 mA (3,5 W)



STM32F103x8 STM32F103xB

Medium-density performance line Arm[®]-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces

Datasheet - production data

Features

- Arm[®] 32-bit Cortex[®]-M3 CPU core
 - 72 MHz maximum frequency, 1.25 DMIPS / MHz (Dhrystone 2.1) performance at 0 wait state memory access
 - Single-cycle multiplication and hardware division
- Memories
 - 64 or 128 Kbytes of Flash memory
 - 20 Kbytes of SRAM
- Clock, reset and supply management
 - 2.0 to 3.6 V application supply and I/Os
 - POR, PDR, and programmable voltage detector (PVD)
 - 4 to 16 MHz crystal oscillator
 - Internal 8 MHz factory-trimmed RC
 - Internal 40 kHz RC
 - PLL for CPU clock
 - 32 kHz oscillator for RTC with calibration
- Low-power
 - Sleep, Stop and Standby modes
 - V_{BAT} supply for RTC and backup registers
- 2x 12-bit, 1 µs A/D converters (up to 16 channels)
 - Conversion range: 0 to 3.6 V
 - Dual-sample and hold capability
 - Temperature sensor
- DMA
 - 7-channel DMA controller
 - Peripherals supported: timers, ADC, SPIs, I²Cs and USARTs
- Up to 80 fast I/O ports
 - 26/37/51/80 I/Os, all mappable on 16 external interrupt vectors and almost all 5 V-tolerant

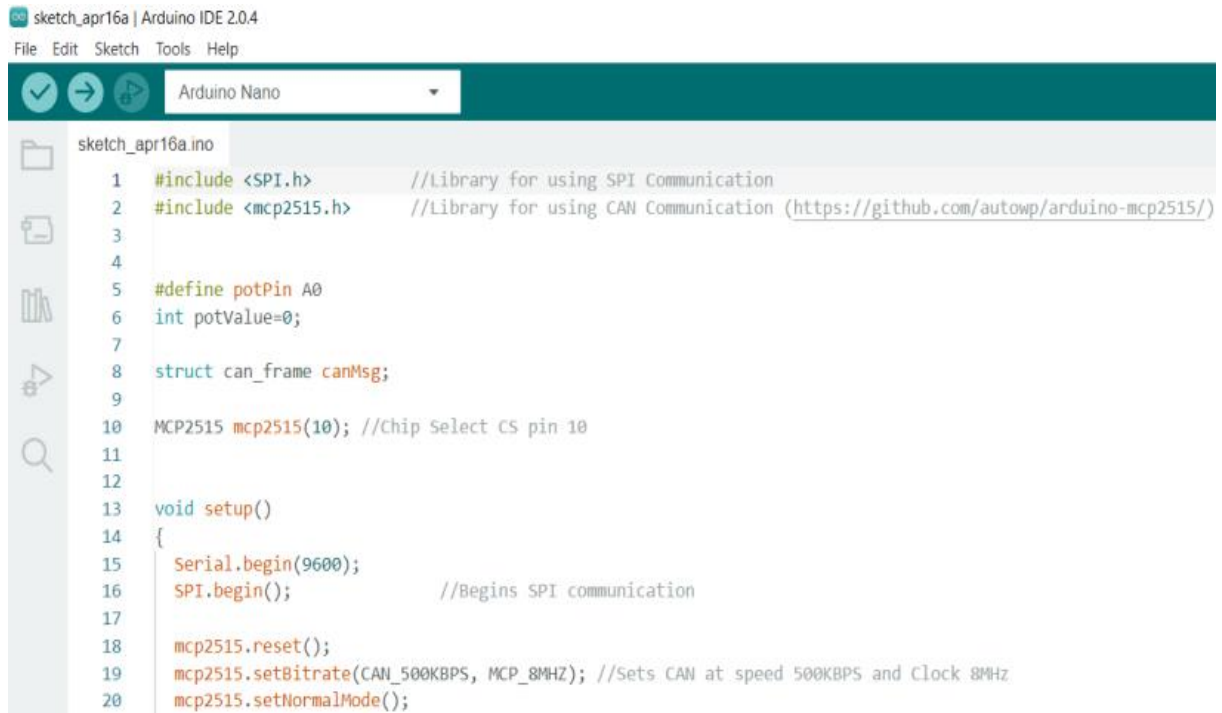


- Debug mode
 - Serial wire debug (SWD) and JTAG interfaces
- Seven timers
 - Three 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input
 - 16-bit, motor control PWM timer with dead-time generation and emergency stop
 - Two watchdog timers (independent and window)
 - SysTick timer 24-bit downcounter
- Up to nine communication interfaces
 - Up to two I²C interfaces (SMBus/PMBus[®])
 - Up to three USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)
 - Up to two SPIs (18 Mbit/s)
 - CAN interface (2.0B Active)
 - USB 2.0 full-speed interface
- CRC calculation unit, 96-bit unique ID
- Packages are ECOPACK[®]

Table 1. Device summary

Reference	Part number
STM32F103x8	STM32F103C8, STM32F103R8 STM32F103V8, STM32F103T8
STM32F103xB	STM32F103RB STM32F103VB, STM32F103CB, STM32F103TB

ANNEXE2



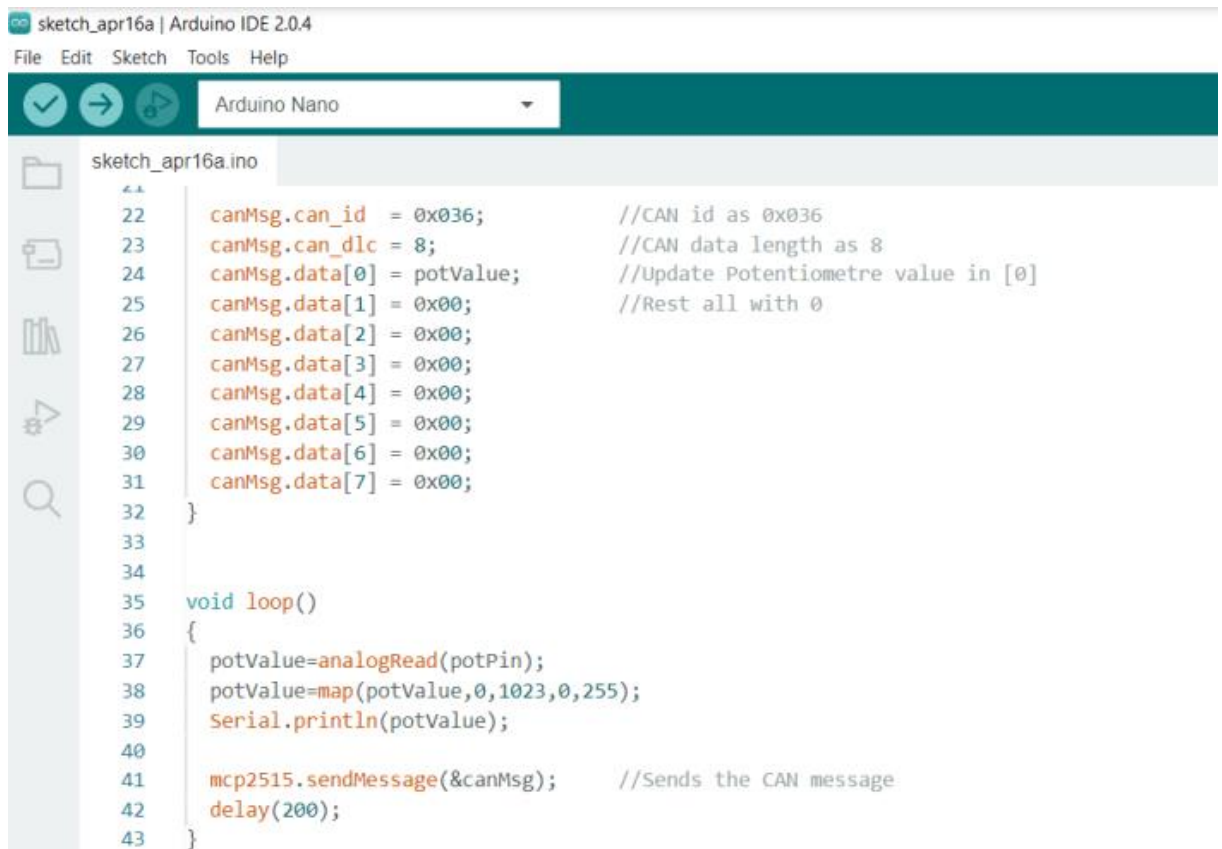
```

sketch_apr16a | Arduino IDE 2.0.4
File Edit Sketch Tools Help

Arduino Nano

sketch_apr16a.ino
1  #include <SPI.h>           //Library for using SPI Communication
2  #include <mcp2515.h>       //Library for using CAN Communication (https://github.com/autowp/arduino-mcp2515/)
3
4
5  #define potPin A0
6  int potValue=0;
7
8  struct can_frame canMsg;
9
10 MCP2515 mcp2515(10); //Chip Select CS pin 10
11
12
13 void setup()
14 {
15   Serial.begin(9600);
16   SPI.begin();           //Begins SPI communication
17
18   mcp2515.reset();
19   mcp2515.setBtrrate(CAN_500KBPS, MCP_8MHZ); //Sets CAN at speed 500KBPS and Clock 8MHz
20   mcp2515.setNormalMode();

```



```

sketch_apr16a | Arduino IDE 2.0.4
File Edit Sketch Tools Help

Arduino Nano

sketch_apr16a.ino
22   canMsg.can_id  = 0x036;           //CAN id as 0x036
23   canMsg.can_dlc = 8;               //CAN data length as 8
24   canMsg.data[0] = potValue;        //Update Potentiometre value in [0]
25   canMsg.data[1] = 0x00;            //Rest all with 0
26   canMsg.data[2] = 0x00;
27   canMsg.data[3] = 0x00;
28   canMsg.data[4] = 0x00;
29   canMsg.data[5] = 0x00;
30   canMsg.data[6] = 0x00;
31   canMsg.data[7] = 0x00;
32 }
33
34
35 void loop()
36 {
37   potValue=analogRead(potPin);
38   potValue=map(potValue,0,1023,0,255);
39   Serial.println(potValue);
40
41   mcp2515.sendMessage(&canMsg);    //Sends the CAN message
42   delay(200);
43 }

```


ANNEXE3

```

1 #include "main.h"
2 #include "stm32f1xx_hal.h"
3 #include "stm32f1xx_hal_gpio.h"
4 #include "stm32f1xx_hal_rcc.h"
5 // Déclaration des adresses MAC source et de destination
6 #define MAC_SOURCE_ADDR {0x00, 0x11, 0x22, 0x33, 0x44, 0x55}
7 #define MAC_DEST_ADDR {0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB}
8
9 // Déclaration des adresses IP source et de destination
10 #define IP_SOURCE_ADDR {192, 168, 1, 100}
11 #define IP_DEST_ADDR {192, 168, 1, 200}
12
13 // Déclaration des registres du W5100
14 #define W5100_SPI_PORT SPI1
15 #define W5100_CS_PIN GPIO_Pin_4
16 #define W5100_CS_PORT GPIOA
17
18
19 CAN_HandleTypeDef hcan;
20
21 void SystemClock_Config(void);
22 static void MX_GPIO_Init(void);
23 static void MX_CAN_Init(void);
24
25 CAN_RxHeaderTypeDef RxHeader;
26
27 uint8_t RxData[8];
28
29
30 int datacheck = 0; //Flag to check if StdId ==0x036
31
32 void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
33
CAN_RxHeaderTypeDef RxHeader;
34
uint8_t RxData[8];
35
36
int datacheck = 0; //Flag to check if StdId ==0x036
37
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
38
{
39     if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO1, &RxHeader, RxData) != HAL_OK)
40     {
41         Error_Handler();
42     }
43
44     if ((RxHeader.StdId == 0x036))
45     {
46         datacheck = 1;
47     }
48 }

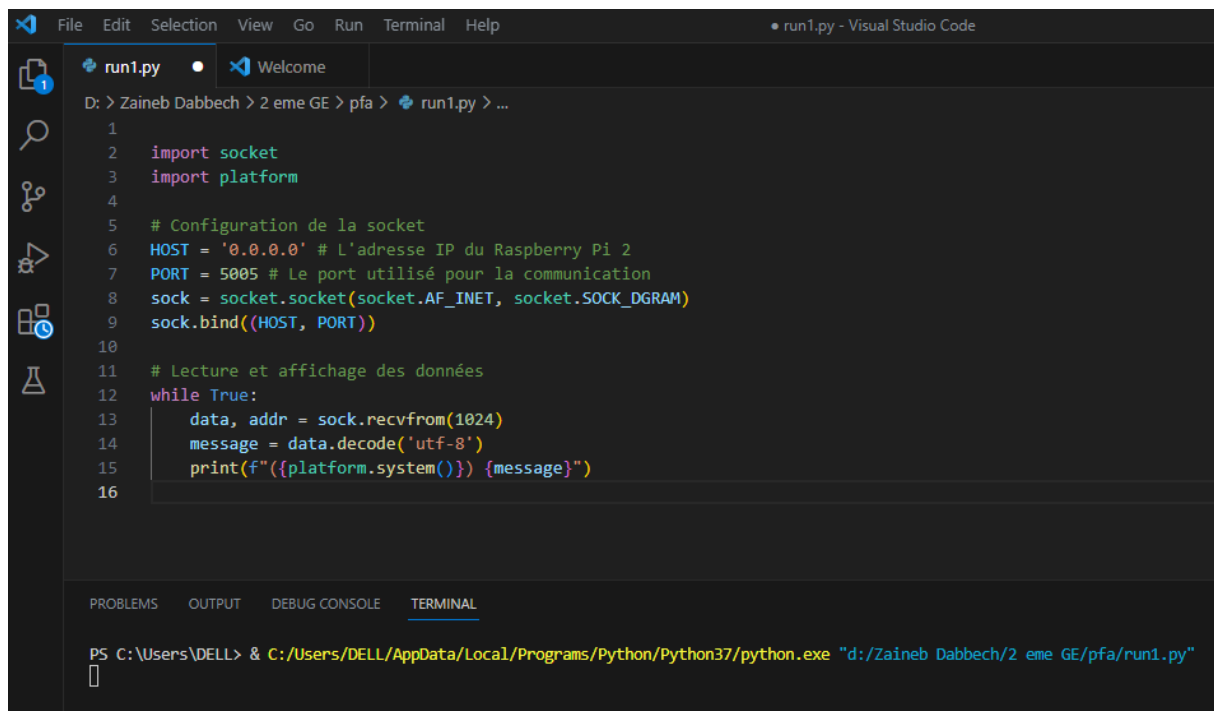
```

```

//
78 // Fonction pour sélectionner le W5100
79 void W5100_Select(void)
80 {
81     GPIO_ResetBits(W5100_CS_PORT, W5100_CS_PIN);
82 }
83
84 // Fonction pour désélectionner le W5100
85 void W5100_Deselect(void)
86 {
87     GPIO_SetBits(W5100_CS_PORT, W5100_CS_PIN);
88 }
89
90 // Fonction d'envoi d'une trame Ethernet via le W5100
91 void W5100_SendEthernetFrame(uint8_t* data, uint16_t length)
92 {
93     W5100_Select();
94
95     // Envoi des données via SPI
96     for (uint16_t i = 0; i < length; i++)
97     {
98         SPI_I2S_SendData(W5100_SPI_PORT, data[i]);
99
100        // Attendre la fin de la transmission
101        while (SPI_I2S_GetFlagStatus(W5100_SPI_PORT, SPI_I2S_FLAG_TXE) == RESET);
102    }
103
104    W5100_Deselect();
105 }
106
107 .....
108 int main(void)
109 {
110
111     HAL_Init();
112     SystemClock_Config();
113     // Initialisation du SPI
114     SPI_Configuration();
115     MX_CAN_Init();
116     HAL_CAN_Start(&hcan);
117
118     if (HAL_CAN_ActivateNotification(&hcan, CAN_IT_RX_FIFO1_MSG_PENDING) != HAL_OK)
119     {
120         Error_Handler();
121     }
122
123     /* USER CODE END 2 */
124
125     /* Infinite loop */
126     /* USER CODE BEGIN WHILE */
127     while (1)
128     {
129
130         if (datacheck)
131         {
132             HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
133         }
134
135
136
137
138 // Configuration des broches CS pour le W5100
139 GPIO_InitTypeDef GPIO_InitStructure;
140 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
141 GPIO_InitStructure.GPIO_Pin = W5100_CS_PIN;
142 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
143 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
144 GPIO_Init(W5100_CS_PORT, &GPIO_InitStructure);
145
146 // Données à envoyer (exemple)
147 uint8_t ethernetFrame[] = {RxHeader.StdId, RxData};
148
149 // Envoi de la trame Ethernet via le W5100
150 W5100_SendEthernetFrame(ethernetFrame, sizeof(ethernetFrame));
151
152

```

ANNEXE4



The image shows a screenshot of the Visual Studio Code editor. The main window displays a Python file named `run1.py` with the following code:

```
1
2 import socket
3 import platform
4
5 # Configuration de la socket
6 HOST = '0.0.0.0' # L'adresse IP du Raspberry Pi 2
7 PORT = 5005 # Le port utilisé pour la communication
8 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9 sock.bind((HOST, PORT))
10
11 # Lecture et affichage des données
12 while True:
13     data, addr = sock.recvfrom(1024)
14     message = data.decode('utf-8')
15     print(f"({platform.system()}) {message}")
16
```

Below the code editor, the TERMINAL panel is active, showing the command used to run the script:

```
PS C:\Users\DELL> & C:/Users/DELL/AppData/Local/Programs/Python/Python37/python.exe "d:/Zaïneb Dabbech/2 eme GE/pfa/run1.py"
```

BIBLIOGRAPHIE

<http://www.datasheet.fr/parts/866258/STM32F103C8-pdf.html>

https://www.researchgate.net/figure/Figure-A3-Arduino-Nano-Data-Sheet-Courtesy-of-element14_fig44_330197859

<https://raspberrypi.fr/raspberry-pi-2/>

<https://slideplayer.fr/slide/9453659/?fbclid=IwAR1ZLcywVkaLHVbaVRudW9DvlcbhlzMI3UCwpJUeaJpTIKWBDwtUuNHvnOQ>

<https://www.youtube.com/watch?v=2gnXKMoFwkc>

https://www.youtube.com/watch?v=QYX_XOjjGOM

<https://www.youtube.com/watch?v=KHNRftBa1Vc>

<https://github.com/autowp/arduino-mcp2515/?fbclid=IwAR31fD4pngONV3YkLkzT1aAXfxRKRYt485WTznHzC7SBBhdTAV6vHf1aaQM>

<https://www.youtube.com/watch?v=JH3cMYErmKI>

<https://www.youtube.com/watch?v=WVgZuJPP6ts>

https://github.com/eziya/STM32F4_HAL_LWIP_LAB/tree/master/STM32F4_HAL_ETH_W5x00_TCPCLIENT?fbclid=IwAR0N1pQIGUvYI9r5Nv_nHmJq_Ft5BonS3qfcCNKoJZdGbe5BgNL_oQJHQPU

https://github.com/autowp/arduino-mcp2515/?fbclid=IwAR0ZPT_wwLiHnvGLNGRgFOQEliVva-MGHg-5cCbJUSSOzH2Nio0eIGuoFU

<https://slideplayer.fr/slide/9453659/?fbclid=IwAR31fD4pngONV3YkLkzT1aAXfxRKRYt485WTznHzC7SBBhdTAV6vHf1aaQM>