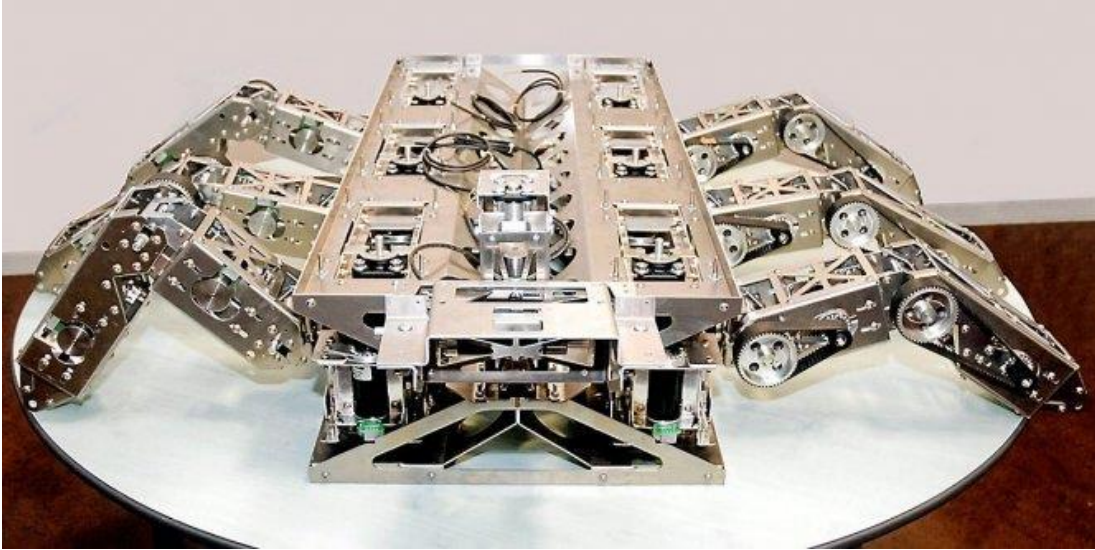


PROJET MEA4

Année 2016-2017



Interfaçage de capteurs CANopen pour la commande des pattes de l'Hexapode R.HEX

Quentin MASSONE

[Rapport Final](#)



SOMMAIRE

I. INTRODUCTION	3
A. PRESENTATION DU ROBOT R.HEX	3
B. LES PATTES.....	4
C. OBJECTIFS DU PROJET.....	5
II. ETUDE DU CODEUR.....	6
A. PRESENTATION	6
B. PROTOCOLE DE COMMUNICATION	9
1. Réseau CAN	9
2. CANopen : la couche application.....	14
a) Présentation.....	14
b) Les différents types d'objets de communication.....	17
(1) Instructions NMT	17
(2) Service Data Object (SDO)	17
(3) Process Data Object (PDO)	20
(4) Layer Setting Service (LSS).....	23
c) Réglages du codeur	23
(1) Les paramètres du bus : vitesse de transmission et numéro de nœud.....	23
(2) Save all bus parameter	23
(3) Save all.....	24
(4) Load factory default	24
d) Communiquer avec le périphérique IXAAT : USB to CAN	24
III. PROGRAMMATION SUR STM32	26
A. CHOIX DE LA CARTE.....	26
B. CONTROLEUR CAN.....	26
1. Transceiver CAN.....	26
2. Ce qu'il faut savoir.....	27
C. DESCRIPTION DU PROGRAMME	32
1. Initialisation du contrôleur	<i>Erreur ! Signet non défini.</i>
IV. RESSOURCES.....	33
A. AVANT-PROPOS	33
B. DOCUMENTS POUR LE CODEUR	33
C. REFERENCES POUR LE CAN	34
D. REFERENCES POUR LE CANOPEN	34
E. DOCUMENTS SUR LA STM32	35

I. Introduction

A. Présentation du robot R.HEX

Le robot R.HEX est un des projets de l'équipe EXPLORE du département robotique du LIRMM. Les missions de l'équipe sont la conception et le développement des outils théoriques et expérimentaux de la robotique mobile pour l'exploration intégrée de l'environnement.

« L'objectif est de créer un prototype de robot hexapode (inspiré du monde des insectes : fourmi), de grande taille, avec de grandes capacités motrices, sensorielles et cognitives. »

Le robot R.HEX pèse 60Kg, fait 1m20 d'envergure et possède quatre degrés de liberté par patte (plus deux pour la tête). Son squelette est composé de 700 pièces, le tout actionné par 27 moteurs (4 par patte).

« Sa principale mission sera d'évoluer dans des environnements fragiles pour des missions d'intervention ou de mesure. Sa structure mécanique, et notamment l'extrémité ponctuelle de ses 6 pattes lui permettront d'avoir un impact minime sur son environnement contrairement à des robots à roues ou à chenilles. Ses différents capteurs et son intelligence embarquée lui fourniront la modélisation en temps réel de son environnement nécessaire à l'élaboration du chemin à parcourir pour réaliser sa tâche. Finalement, sa redondance lui apportera une agilité à se mouvoir et sa capacité de charge rendra possible l'apport de matériel ou le transport de déchets. »

A l'heure actuelle, la structure du robot est achevée et des simulations ont été faites. Un terrain virtuel en 3D a été créé pour faire évoluer le robot dans un environnement proche de la réalité. Cela a permis de tester diverses sortes de marches, d'automatiser des comportements et d'affiner les algorithmes d'apprentissage du prototype. Cette démarche permet d'aller beaucoup vite qu'en extérieur.

Mon projet porte sur la partie électronique du robot qui est toujours en cours. En effet, mon rôle est de travailler sur la commande des pattes du robot.

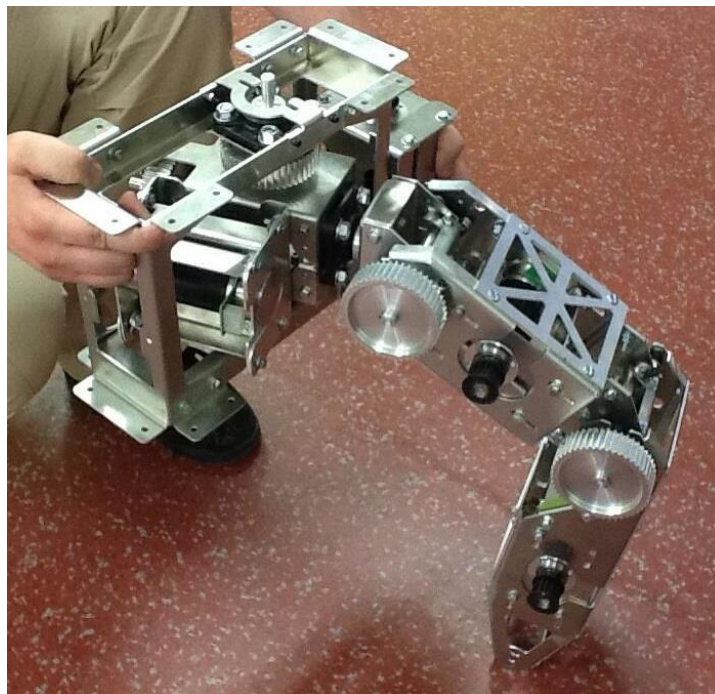
B. Les pattes

Le robot dispose de 6 pattes équipées de 4 moteurs permettant ainsi 4 rotations.

Pour une commande optimale du robot, il est nécessaire de connaître sa position à tout instant et donc la position exacte des pattes.

Il faut donc connaître précisément les 4 angles résultant des 4 rotations des moteurs. La solution choisie est d'utiliser 4 codeurs qui renverront les angles exacts des rotations, voir les vitesses de rotation. Cependant, au vu des faibles amplitudes de mouvements d'un robot comme celui-là, il est fort possible que les vitesses ne soient pas intéressantes.

Avec ces 4 angles (et éventuellement ces 4 vitesses), il est alors possible d'asservir les 4 moteurs présents sur chaque patte.



Patte du robot R.HEX

C. Objectifs du projet

Ma mission majeure est de comprendre comment fonctionnent les codeurs utilisés par le robot. Il faut donc se documenter sur ses caractéristiques et son protocole de communication.

Une fois que cela est fait il faut à l'aide d'un logiciel essayer de communiquer avec le codeur. Cela permet de vérifier que l'on a bien compris quelle trame il faut envoyer.

Puis, comme le but est d'utiliser le codeur sur un robot, il faut pouvoir récupérer les informations à l'aide d'un microcontrôleur. On doit donc réaliser un programme permettant la communication et la récupération des données.

Enfin, quand le programme marche pour un codeur, on peut passer à plusieurs codeurs reliés au même microcontrôleur.

II. Etude du codeur

A. Présentation

Le codeur choisi est un codeur absolu monotour Sendix M3658 de Kübler avec capteur magnétique. « Absolu » signifie que l'angle est exprimé par rapport à une position de référence. « Monotour » indique qu'il n'est pas possible de connaître le nombre de tours fait par le codeur par rapport à la position de référence contrairement à un « multi-tours ». De plus, ce codeur renvoie un angle non signé compris entre 0 et 360°.



Le codeur est basé sur le réseau CAN avec interface CANopen (couche application). La partie suivante de ce rapport est consacrée au protocole.

Sans rentrer dans les détails techniques du codeur, celui-ci dispose de 5 fils en sortie : 0V, V+, CAN_GND, CAN_HIGH, CAN_LOW. Nous reviendrons sur leurs significations.

La résolution de l'angle est sur 16 bits et non 14 bits comme indiqué dans la documentation technique (valeurs de l'angle : 1-65536 et non 1-16384).

Sa plage de mesure est de 360° avec une précision annoncée à $\pm 1^\circ$. Il a semblé toutefois que la précision remarquée lors des tests était meilleure.

Sa vitesse de rotation maximale est 6000 tr/min.

Enfin sa tension d'alimentation est comprise dans une plage de 8-25V avec une consommation de 25 mA.

Pour de plus amples informations sur les caractéristiques du codeur, il suffit de lire la documentation technique que l'on peut trouver sur le site de Kuebler. En voici quelques extraits :

Caractéristiques mécaniques	
Vitesse de rotation max.	6000 min ⁻¹
Couple de démarrage - à 20°C [68°F]	< 0.06 Nm
Charge admissible sur l'arbre	radiale 40 N axiale 20 N
Poids	env. 0.2 kg [7.06 oz]
Protection selon EN 60529/DIN 40050-9	IP67 / IP69k
Plage de températures de travail	-40°C ... +85°C [-40°F ... +185°F]
Matières	arbre sortant / creux acier inoxydable bride aluminium boîtier zinc moulé sous pression câble PUR
Résist. aux chocs selon EN 60068-2-27	5000 m/s ² , 6 ms
Résist. aux vibrations selon EN 60068-2-6	300 m/s ² , 10 ... 2000 Hz
Chocs permanents selon EN 60068-2-27	1000 m/s ² , 2 ms
Vibration (bruit à large bande) selon EN 60068-2-64	5 ... 2500 Hz, 100 m/s ² - rms

Caractéristiques électriques	
Tension d'alimentation	8 ... 30 V DC
Consommation (sans charge)	max. 25 mA
Protection contre les inversions de polarité de la tension d'alimentation	oui
Plage de mesure	360°
Précision absolue, 25°C [77°F]	±1°
Répétabilité, 25°C [77°F]	±0.2°
Actualisation des données	400 µs
Conforme aux normes CE selon	Directive CEM 2014/30/EU Directive RoHS 2011/65/UE

LED de diagnostic (bicolore, rouge/vert)	
LED fixe ou clignotante	rouge Signalisation de défaut verte Signalisation d'état

Caractéristiques des interfaces CANopen	
Résolution	ERREUR 1 ... 16384 (14 bits), facteur d'échelle défaut: 16384 (14 bits)
Code	binaire
Interface	CAN high-speed selon ISO 11898, Basic-CAN et Full-CAN, Spécification CAN 2.0 B
Protocole	Profil CANopen DS406 V3.2 avec compléments spécifiques au constructeur, Service LSS DS305 V2.0

Vitesse de transmission	10 ... 1000 kbit/s réglable par logiciel
Adresse de nœud	1 ... 127 réglable par logiciel
Terminaison	réglable par logiciel
Protocole LSS	CIA LSS Protocole DS305, Support d'instructions global pour l'adresse de nœud et la vitesse de transmission, Instructions sélectives grâce aux attributs de l'objet Identity

Pour trouver la documentation, il suffit de se rendre sur la page des codeurs absolus monotour puis choisir le codeur M3658.

← → 🔒 Sécurisé | <https://www.kuebler.com/francais/index.html> 🔍 ☆ 🌐

🇫🇷 | more languages 📺 📱 📧 📧 Login.

Google™ Benutzerdefinierte Suche 🔍

Kübler

🏠 **Société**
l'entreprise

Produits
produits & solutions

Nouveautés
nouveau 2017

Presse
imprimés & médias

Service
support & téléchargement

Contact
e-mail & adresse


Produits Kübler

- ▶ Technologie capteurs
 - Codeurs incrémentaux
 - Codeurs absolus monotour** →
 - Codeurs absolus multitours
 - Codeurs Sans roulement
 - Mesure linéaire
 - Inclinomètres
 - Connectique
 - Accessoires
- ▶ Technologie compteurs
- ▶ Technique des process
- ▶ Technique de sécurité
- ▶ Collecteurs tournants
- ▶ Technique de transmission

Produits Framo Morat

- ▶ Engrenages
- ▶ Vis sans fin

Modbus **CANopen** **Analog**




← → 🔒 Sécurisé | <https://www.kuebler.com/k2014/j/fr/produkte/list/Sensortechnik/Rotativ/Absolute/Singleturn> 🔍 ☆ 🌐

▶ technique des process

Solutions


- ▶ Solutions par branches
- ▶ Sécurité fonctionnelle
- ▶ Interface BiSS
- ▶ Produits & Systèmes OEM

Sendix 3651




Analogique 4...20 mA
Analog 0...5 V
Analogique 0...10 V
Dimension 36 mm
Résolution:
max: 12 bit | Monotour
Arbre max. 8 mm avec surface

Sendix 3671




Analogique 4...20 mA
Analog 0...5 V
Analogique 0...10 V
Dimension 36 mm
Résolution:
max: 12 bit | Monotour
Arbre max. 10 mm

Sendix M3658 CANopen →



CANopen
Dimension 36 mm
Résolution:
max: 14 bit | Monotour
Arbre max. 8 mm avec surface

Sendix M3678 CANopen



CANopen
Dimension 36 mm
Résolution:
max: 14 bit | Monotour
Arbre max. 10 mm

Une fois que l'on a trouvé la page relative à notre codeur, il suffit d'aller dans la partie fiche technique. Voici le lien de la page relative à notre codeur :

https://www.kuebler.com/k2014/j/fr/produkte/details/drehgeber/Absolut_Singleturn/Welle//Kompakt/M3658_CANopen

Voici le lien du document :

https://www.kuebler.com/k2016/pdf?M3658-M3678_CANopen_fr.pdf

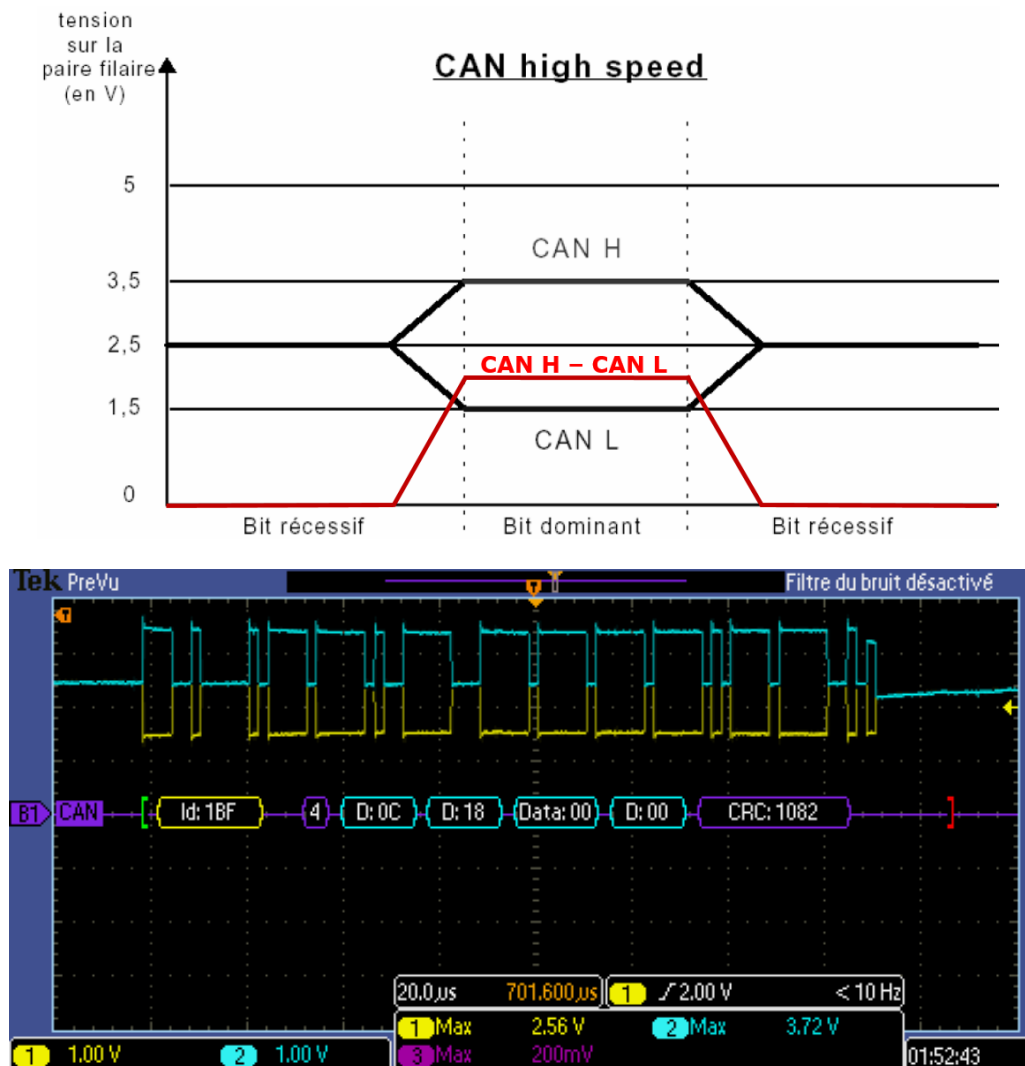
B. Protocole de communication

1. Réseau CAN

Mon but dans cette partie n'est pas de faire un cours sur le réseau mais plutôt d'énoncer ce qui est important pour la suite.

Un réseau CAN est donc un réseau série composé de 2 fils de données : CAN HIGH et CAN LOW. Les états logiques sont codés par différence de potentiel entre les 2 fils.

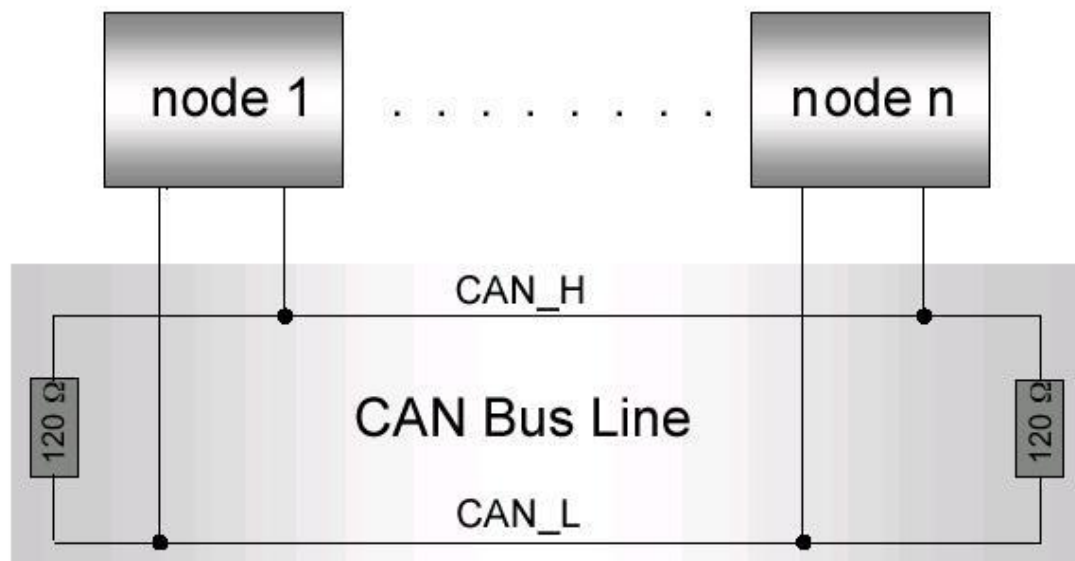
Il existe 2 configurations de bus : CAN low speed (jusqu'à 125Kb/s) et CAN high speed (125Kb/s - 1Mb/s). Suivant la configuration, les tensions des fils varient. Notre codeur fonctionne en CAN high speed avec une vitesse de transmission réglable allant de 10Kb/s à 1Mkb/s. Voici les tensions des fils que j'ai pu vérifier à l'oscilloscope.



Voici une trame CAN envoyée par le codeur que j'ai pu récupérer sur un oscilloscope équipé d'un interpréteur CAN. En jaune c'est CAN_LOW et en cyan c'est CAN_HIGHT.

Dans une trame CAN, on parle de **bit récessif** lorsque la différence des tensions est de **0V** ; il correspond à un **1 logique**. On parle de **bit dominant** quand on a **2V** ; il correspond à un **0 logique**.

Voici donc à quoi ressemble un réseau CAN avec un appareil à chaque nœud. Un réseau CAN en high speed permet d'avoir jusqu'à 30 nœuds sur le même réseau.



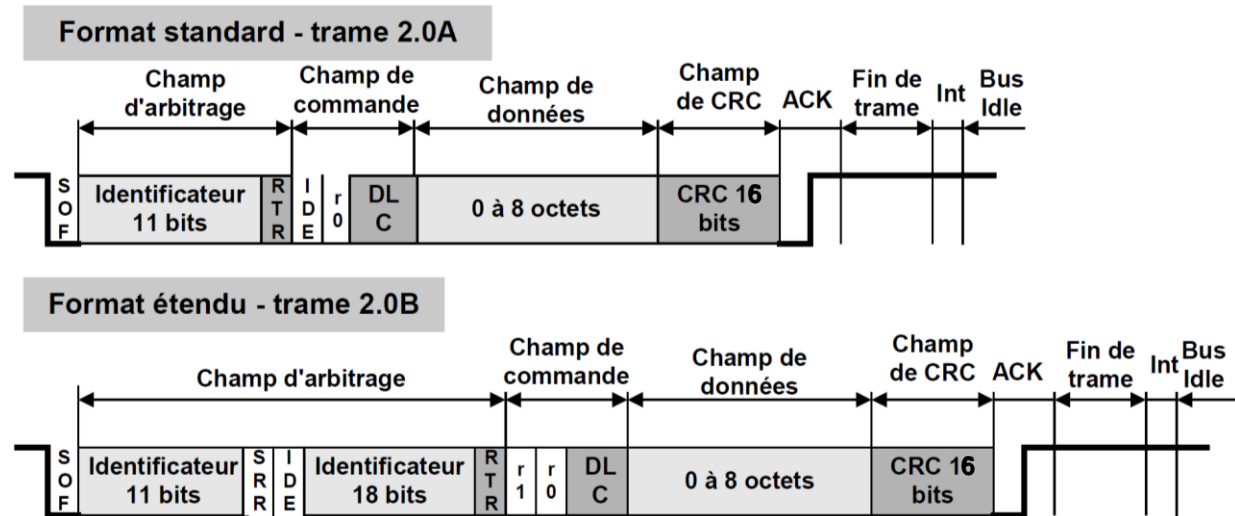
La spécification CAN V2.0 comprend 2 versions : CAN 2.0.A et CAN 2.0.B

CAN 2.0.A correspond au format de trame standard avec un identifiant codé sur **11 bits**. **CAN 2.0.B** correspond au format de trame étendue avec un identifiant codé sur **29 bits**. CAN 2.0.B permet toutefois un identifiant sur 11 bits. C'est d'ailleurs le cas de notre codeur puisqu'il dispose d'une spécification CAN 2.0.B avec un identifiant uniquement sur 11 bits.

Il existe plusieurs types de trame :

- **Data Frame** : trames transportant des données d'un producteur vers des consommateurs, sans garantie de traitement.
- **Remote Frame** : trames de requête en polling émises par un maître vers un ou plusieurs esclaves, pour demander le renvoi d'une trame de données.
- **Error Frame** : trames émises lorsqu'une station détecte une erreur de transmission sur le bus.
- **Overload Frame** : trames émises pour demander un laps de temps supplémentaire entre des trames (de données ou de requête) successives.

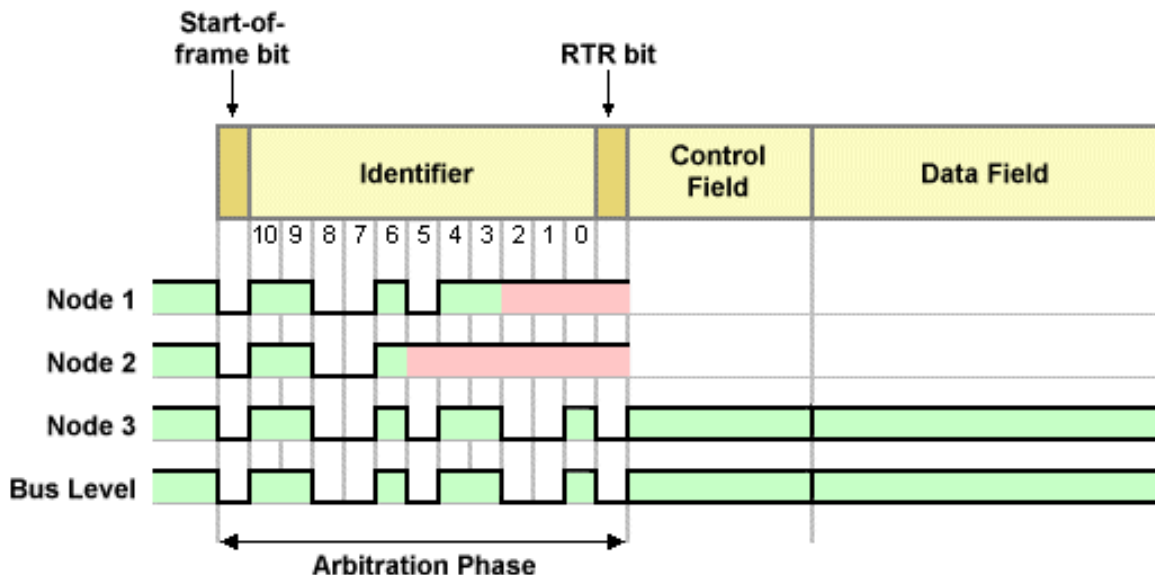
Voyons tout d'abord la structure généralisée d'une trame :



SOF : start of frame, 1 bit dominant (0 logique)

Champ d'arbitrage

Il est appelé ainsi car cette partie de la trame définit la priorité du message. En effet, si plusieurs messages provenant de nœuds différents sont émis en même temps, il faut pouvoir définir lequel des messages sera émis. Voici un schéma qui montre qu'un bit dominant (0 logique) est prioritaire par rapport à un bit récessif (1 logique).



Plus l'identifiant est petit, plus le message est prioritaire.

Le **bit RTR** (Remote Transmission Request) indique si c'est une trame de requête ou de donnée : bit dominant pour trame de donnée, bit récessif pour trame de requête. Une trame de donnée est donc plus prioritaire qu'une trame de requête.

Le **bit SRR** (substitute remote request) présent dans le format étendu est toujours récessif. Il permet à une trame au format standard d'être plus prioritaire qu'une trame au format étendu si jamais les 11 premiers bits sont identiques.

Le **bit IDE** (Identifier Extension bit) présent dans le format étendu est récessif ce qui signifie que c'est une trame au format étendu.

Champ de commande

Le **bit IDE** présent dans le format standard est dominant et indique que c'est une trame au format standard.

R0 présent dans le format standard est un bit de réserve mais doit être dominant. **R1** et **R0** présents dans le format étendu sont des bits de réserve.

DLC (Data Length Code) est sur 4 bits et indique le nombre d'octets du champ de données.

Champ de donnée

Cette partie de la trame contient les données de la trame pouvant aller de 0-8 octets.

Champ de CRC

Ce champ de vérification des données est composé de 2 parties :

Le code de vérification des données transmises sur 15 bits. Le récepteur compare son code à celui de l'émetteur. S'il y a une différence alors présence d'erreurs donc pas d'acquiescement.

Un bit délimiteur de vérification de données toujours à l'état 1 (récessif) qui marque la fin de vérification.

ACK

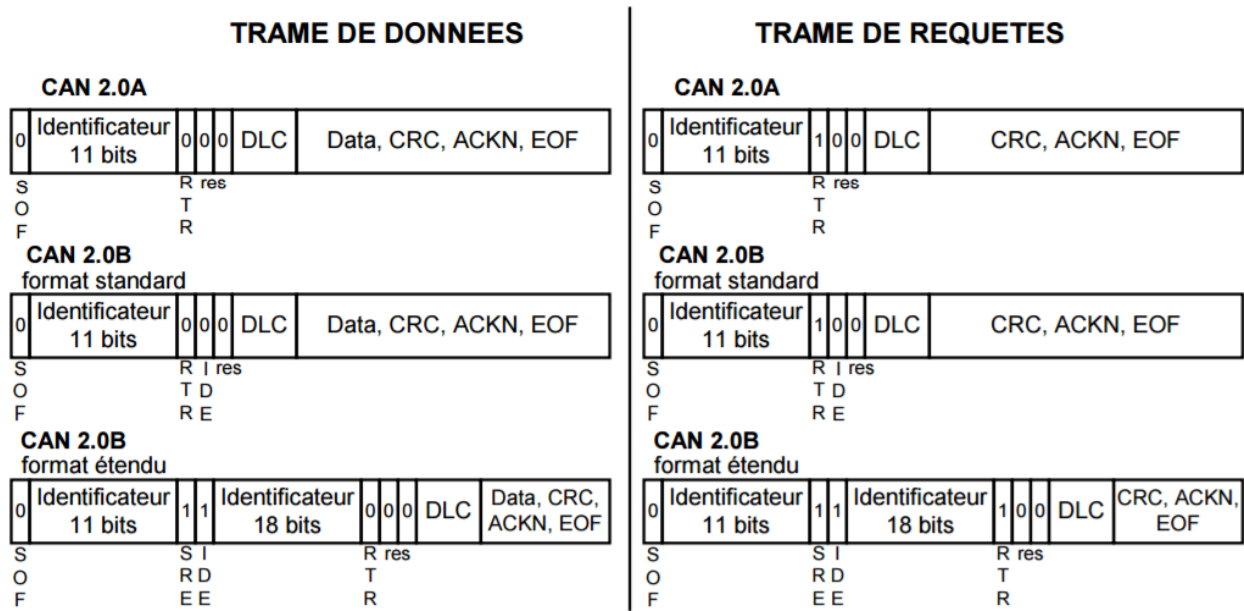
Ce champ d'acquiescement est composé de 2 bits. Un bit d'acquiescement qui est dominant quand le calcul du code de vérification est correct ; récessif quand il ne l'est pas. Un bit délimiteur d'acquiescement, toujours récessif.

Fin de trame : Suite de 7 bits récessifs qui indiquent la fin de la trame.

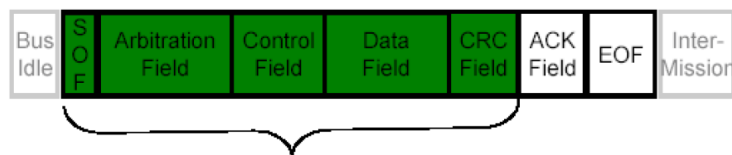
Intertrame : Suite de 3 bits récessifs qui séparent 2 trames consécutives.

Les trames qui vont nous intéresser par la suite sont les trames de données et de requête.

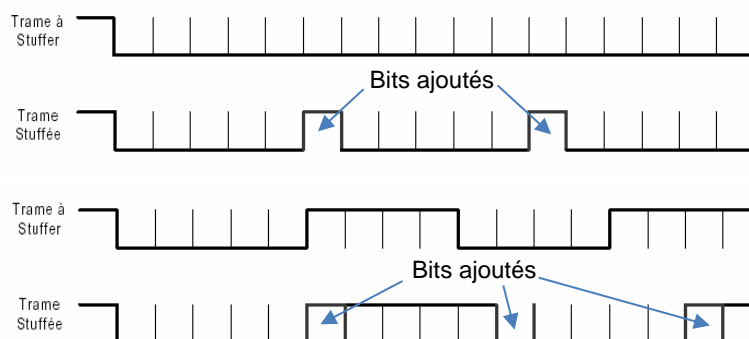
Les trames de données sont structurées de la même manière que précédemment. Les trames de requête ne comportent pas de champ de données (donc DLC = 0) et le bit RTR doit être récessif.



Un dernier point important à aborder est le bit-stuffing. En effet, après 5 bits de même niveau, un bit (sans signification) de niveau inverse est ajouté.



Bit-Stuffing sur ces champs uniquement



2. CANopen : la couche application

a) Présentation

CANopen est un protocole de communication qui s'appuie sur la couche application du réseau CAN (CAL : CAN Application Layer). Le profil de communication est clairement défini par l'organisation CIA (CAN In Automation) sous le nom de CIA DS301. Le codeur supporte le profil de communication CANopen DS301 V4.2 que l'on peut trouver sur le site de l'organisation CIA.

<https://www.can-cia.org/standardization/specifications/>

Kuebler fournit 2 documents très utiles sur le protocole CANopen que l'on peut trouver sur la page du codeur dans la partie documentation :

https://www.kuebler.com/k2014/j/fr/produkte/details/drehgeber/Absolut_Singleturn/Welle//Kompakt/M3658_CANopen

Le document « Programming instruction: CANopen » fournit des exemples de trames et explique les différents types d'objet de communication.

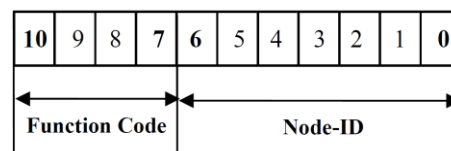
Le document « Manuel: M36xx CANopen Singleturn » est le document de référence du codeur en ce qui concerne le protocole CANopen. Il contient notamment le dictionnaire d'objets.

Dictionnaire d'objets

Pour communiquer, le protocole se base sur un dictionnaire d'objets stockés dans la mémoire EEPROM du codeur. Ils sont accessibles via un index sur 2 octets et éventuellement un sous-index sur 1 octet. Ce dictionnaire contient l'ensemble des paramètres qui décrivent le codeur et son comportement vis-à-vis du réseau.

Identifiant COB

Il correspond à l'identifiant CAN sur 11 bits nécessaire pour envoyer un message. Il se compose d'un code de fonction à 4 bits et d'un numéro de nœud à 7 bits.



Le numéro de nœud doit être unique pour chaque codeur du même réseau et sa valeur peut aller de 0x01 à 0x7F (1 à 127 en décimal).

Tous les identifiants sont définis dans le répertoire d'objets par des valeurs standard. Ces identifiants peuvent cependant être modifiés par accès SDO.

Le code de fonction va spécifier quel type d'objet de communication (COB) est envoyé et précise le sens du message en prenant comme référentiel le codeur. On parle de réception quand le codeur reçoit un objet ; transmission quand il en transmet un.

Voici la liste des différents types d'objet de communication avec leur identifiant respectif.

Objet de communication	COB-ID(s) hex	COB-ID(s) bin	Sens du message
NMT node control	000	000 0000 0000	Réception
SYNC	080	000 1000 0000	Réception
Emergency	080 + NodeID	000 1(NodeId)	Transmission
TimeStamp	100	001 0000 0000	Réception
PDO	180 + NodeID	001 1(NodeId)	Transmission PDO1
	200 + NodeID	010 0(NodeId)	Réception PDO1
	280 + NodeID	010 1(NodeId)	Transmission PDO2
	300 + NodeID	011 0(NodeId)	Réception PDO2
	380 + NodeID	011 1(NodeId)	Transmission PDO3
	400 + NodeID	100 0(NodeId)	Réception PDO3
	480 + NodeID	100 1(NodeId)	Transmission PDO4
SDO	580 + NodeID	101 1(NodeId)	Transmission
	600 + NodeID	110 0(NodeId)	Réception
NMT node monitoring (node guarding / heartbeat)	700 + NodeID	111 0(NodeId)	Transmission
LSS	7E4	111 1110 0100	Transmission
	7E5	111 1110 0101	Réception

Les 4 bits du code de fonction sont en bleu

Comme on peut le constater sur le tableau, certains identifiants ne nécessitent pas de nœud, ceux sont des objets broadcast. Ils ont une priorité haute.

Nous reviendrons sur la plupart de ces objets de communication.

En ce qui concerne la priorité des messages, on conserve la règle du réseau CAN sur la priorité des messages. Plus l'identifiant du message est bas, plus le message est prioritaire.

Voici le récapitulatif de tous les types d'objet de communication. Le premier des deux tableaux contient les objets broadcast.

object	function code (binary)	resulting COB-ID	Communication Parameters at Index
NMT	0000	0	-
SYNC	0001	128 (80h)	1005h, 1006h, 1007h
TIME STAMP	0010	256 (100h)	1012h, 1013h

object	function code (binary)	Resulting COB-IDs	Communication Parameters at Index
EMERGENCY	0001	129 (81h) – 255 (FFh)	1014h, 1015h
PDO1 (tx)	0011	385 (181h) – 511 (1FFh)	1800h
PDO1 (rx)	0100	513 (201h) – 639 (27Fh)	1400h
PDO2 (tx)	0101	641 (281h) – 767 (2FFh)	1801h
PDO2 (rx)	0110	769 (301h) – 895 (37Fh)	1401h
PDO3 (tx)	0111	897 (381h) – 1023 (3FFh)	1802h
PDO3 (rx)	1000	1025 (401h) – 1151 (47Fh)	1402h
PDO4 (tx)	1001	1153 (481h) – 1279 (4FFh)	1803h
PDO4 (rx)	1010	1281 (501h) – 1407 (57Fh)	1403h
SDO (tx)	1011	1409 (581h) – 1535 (5FFh)	1200h
SDO (rx)	1100	1537 (601h) – 1663 (67Fh)	1200h
NMT Error Control	1110	1793 (701h) – 1919 (77Fh)	1016h, 1017h

La dernière colonne des tableaux indique l'index des objets de communications dans le dictionnaire d'objet.

b) Les différents types d'objets de communication

Comme nous l'avons vu dans le tableau précédent il existe plusieurs types d'objet de communication qui sont décrits par des services et des protocoles. CANopen fait appel à quatre objets de communication ayant des caractéristiques distinctes :

- Objets de données process (PDO) pour des transferts de données en temps réel.
- Objets de données service (SDO) pour la transmission de paramètres et de programmes.
- Gestion du réseau : NMT, Life-Guarding, Heartbeat.
- Objets prédéfinis pour la synchronisation, l'horodatage, les cas d'urgence : SYNC, TimeStamp, Emergency.

Seul les objets de communications NMT, SDO, PDO, SYNC et LSS seront traités car les autres n'ont pas été utilisés.

(1) Instructions NMT

Ces instructions permettent la gestion du réseau. En effet, on définit le mode de fonctionnement du codeur en lui transmettant des objets NMT. L'identifiant de ces objets est 0. C'est donc une communication Broadcast et la priorité est la plus élevée possible. L'objet NMT est composé de 2 octets : le premier contient l'instruction et le second le numéro de nœud.

Identifiant	Octet 0	Octet 1
000h	Instruction	Numéro de nœud

D'un point de vue CAN, la trame a pour identifiant l'ID-COB, DLC égal 2 et le champ de donnée est composé des 2 octets. Ce sera la même logique pour tous les objets de communication. Voici les différentes instructions possibles :

Instruction	Description	Mode
01h	Start Remote Node	Operational mode
02h	Stop Remote Node	Pre-operational mode
80h	Enter Pre-Operational	Pre-operational mode
81h	Reset Node	
82h	Reset Communication	

(2) Service Data Object (SDO)

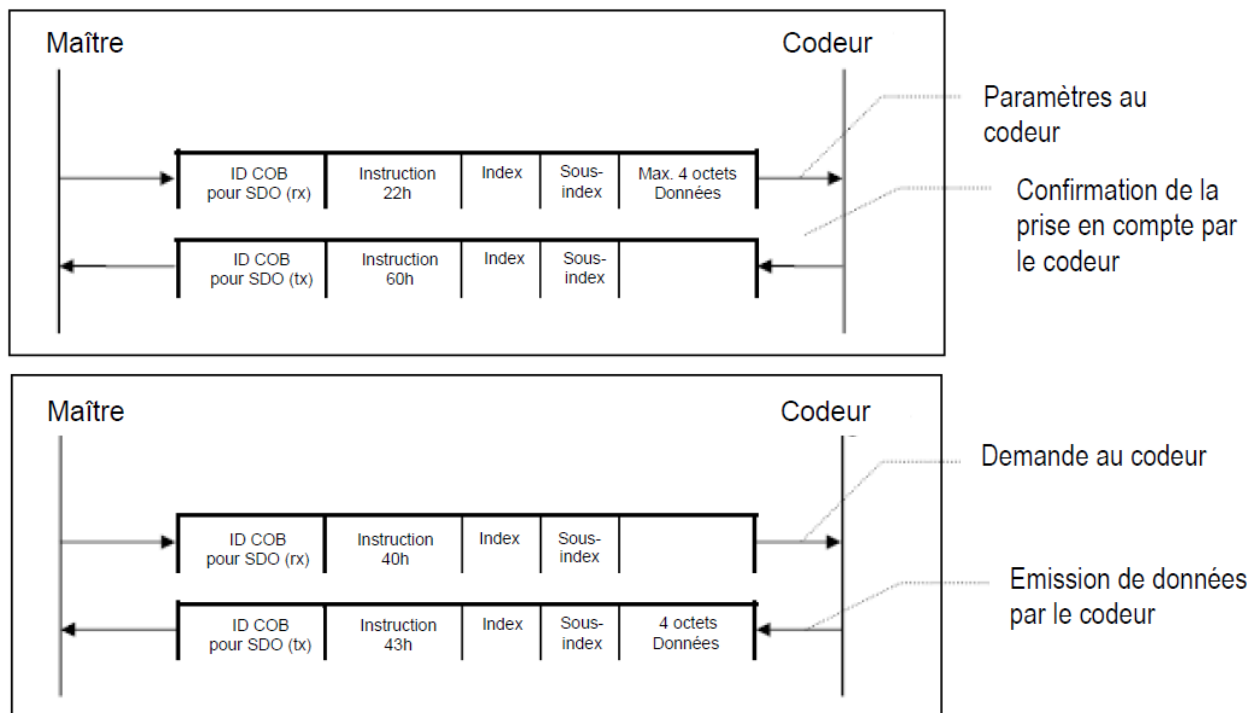
Ces objets de service permettent un accès direct en lecture et en écriture au dictionnaire d'objets. Toutefois, certains ne sont accessibles qu'en lecture.

Un objet se caractérise par un index (et parfois un sous-index) pour y accéder et par une valeur. Chaque objet dispose d'une valeur par défaut établit par le constructeur. La liste des objets du codeur se trouve dans le manuel CANopen fournit par Kuebler.

Pour émettre et recevoir un objet, on dispose de 2 COB-ID :

- SDO (tx) (Codeur→Maître) : 580h (1408) + Numéro de noeud
- SDO (rx) (Maître→Codeur) : 600h (1536) + Numéro de noeud

Exemple d'écriture dans un objet puis d'une demande de lecture d'un objet :



Pour transmettre l'objet, on précise tout d'abord son identifiant qui est 600h+Numéro_de_noeud.

Puis, il y a un octet d'instruction. Celui-ci indique si l'on veut écrire dans un objet (il précise la taille de la valeur de l'objet) ou si l'on veut le lire.

Il y a ensuite l'index sur 2 octets (en premier on a l'octet de poids faible) suivi du sous-index sur 1 octet. Ce sous-index est à 00h si jamais l'objet en question n'a pas de sous-index.

Enfin, il y a la valeur de l'objet dont la taille varie de 0 à 4 octets. Les octets de cette valeur sont écrits du poids le plus faible vers le plus fort.

Voici donc la structure de la trame :

Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5	Octet 6	Octet 7
Instruction	Index LSB	Index MSB	Sous- index	Data LSB -----> Data MSB			

Lors d'une écriture, l'instruction précise la taille de la donnée qui peut aller de 1 à 4 octets. Le DLC du CAN peut donc aller de 5 à 8. Lors d'une demande de lecture, l'instruction est égale à **40h** et l'objet de communication SDO que l'on transmet ne transporte pas de donnée. Par conséquent DLC = 4 (1 octet d'instruction + 2 octets d'index + 1 octet de sous-index).

L'objet de communication transmis par le codeur est nécessairement de 8 octets. L'instruction de cette réponse peut prendre 2 valeurs :

- 60h qui indique que l'objet de communication reçu par le codeur est correct.
- 80h qui signifie qu'il y a une erreur dans l'objet reçu.

Si c'est une écriture, chacun des 4 octets de données de l'objet transmis par le codeur est égal à 00h (s'il n'y a pas d'erreur dans l'objet reçu) ou égales à une certaine valeur qui dépend de la nature de l'erreur. Si c'est une demande de lecture, les 4 octets de données de l'objet transmis par le codeur contiennent l'objet demandé (s'il n'y a pas d'erreur dans l'objet reçu). S'il y a une erreur c'est la même chose qu'avec l'écriture.

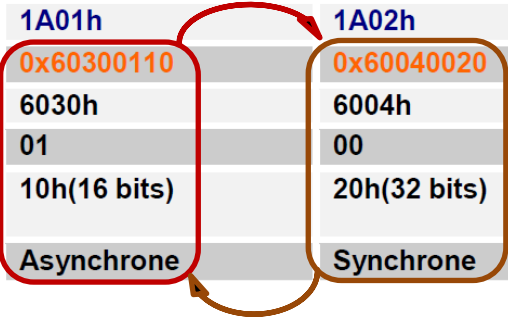
Voici le tableau regroupant la liste des instructions possibles.

Instruction (Expedited Protocol)	Type	Fonction
22h	SDO(rx), Initiate Download Request	Envoi des paramètres au codeur (longueur max. des données 4 octets)
23h	SDO(rx), Initiate Download Request	Envoi des paramètres au codeur (longueur des données 4 octets)
2Bh	SDO(rx), Initiate Download Request	Envoi des paramètres au codeur (longueur des données 2 octets)
2Fh	SDO(rx), Initiate Download Request	Envoi des paramètres au codeur (longueur des données 1 octet)
60h	SDO(tx), Initiate Download Response	Confirmation de la prise en compte au maître
40h	SDO(rx), Initiate Upload Request	Demande des paramètres du codeur
43h	SDO(tx), Initiate Upload Response	Paramètres au maître, longueur des données = 4 octets (unsigned 32)
4Bh	SDO(tx), Initiate Upload Response	Paramètres au maître, longueur des données = 2 octets (unsigned 16)
4Fh	SDO(tx), Initiate Upload Response	Paramètres au maître, longueur des données = 1 octet (unsigned 8)
80h	SDO(tx), Abort Domain Transfer	Le codeur envoie un code d'erreur au maître

(3) Process Data Object (PDO)

Ces objets de communication sont utilisés pour l'échange de données en temps réel. Il existe 3 PDO de transmission (TPDO) et 3 PDO de réception (RPDO). Je n'ai pas utilisé les RPDO, par conséquent je ne m'attacherai qu'à décrire les TPDO. Lors de mon travail, j'ai pu me rendre compte qu'il existe 2 autres PDO, PDO4 et PDO5. Mais comme ils ne sont pas énoncés dans le manuel CANopen du codeur, je n'en parlerai pas. Voici d'ailleurs les 3 TPDO utilisables avec leurs valeurs par défaut :

Mappage	TPDO1 1800h	TPDO2 1801h	TPDO3 1802h
Objet mappé	1A00h	1A01h	1A02h
Enregistrement	0x60040020	0x60300110	0x60040020
Objet	6004h	6030h	6004h
Sous-index	00	01	00
Longueur des données	20h(32 bits)	10h(16 bits)	20h(32 bits)
	Asynchrone	Asynchrone	Synchrone



Nous allons nous servir de ce tableau dans les explications qui suivent.

Un PDO est décrit par 2 objets dans le dictionnaire d'objet :

- Le PDO Communication Parameter, qui indique comment est transmis l'objet. L'index de cet objet prend les valeurs 1800h, 1801h et 1802h dans le tableau.
- Le PDO Mapping Parameter qui indique quelles sont les données transportées. L'index de cet objet prend les valeurs 1A00h, 1A01h et 1A02h dans le tableau.

La première ligne Mappage dans le tableau correspond donc aux index des PDO Communication Parameter.

La seconde ligne Objet mappé contient les index des PDO Mapping Parameter.

La troisième ligne Enregistrement indique la donnée transportée. On a le détail de cette donnée dans les 3 lignes qui suivent :

- L'objet à l'index 6004h, sous-index 00h correspond à la position du codeur.
- L'objet à l'index 6030h, sous-index 01h correspond à la vitesse du codeur.

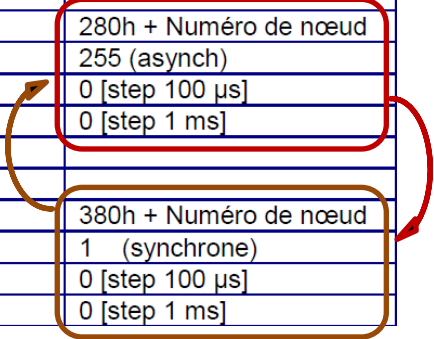
La dernière ligne précise comment est demandé l'objet. Cette information est contenue dans le PDO Communication Parameter sur lequel nous allons revenir.

Il faut noter que ce tableau est extrait du manuel CANopen du codeur et présente une grosse erreur. **En effet, ce n'est pas la vitesse qui est mappée dans l'objet 1A01h mais la position. La vitesse est mappée dans l'objet 1A02h.**

Le PDO Communication Parameter

Cet objet indique comment est transmis le PDO. Les informations de cet objet sont répartis sur plusieurs sous-index. Le tableau suivant contient les valeurs par défaut.

1800h	TPDO1 Communication Parameter	
01h	COB-ID	180h + Numéro de nœud
02h	Transmission Type	255 (asynch)
03h	Inhibit Time	0 [step 100 µs]
05h	Event timer	0 [step 1 ms]
1801h	TPDO2 Communication Parameter	
01h	COB-ID	280h + Numéro de nœud
02h	Transmission Type	255 (asynch)
03h	Inhibit Time	0 [step 100 µs]
05h	Event timer	0 [step 1 ms]
1802h	TPDO3 Communication Parameter	
01h	COB-ID	380h + Numéro de nœud
02h	Transmission Type	1 (synchrone)
03h	Inhibit Time	0 [step 100 µs]
05h	Event timer	0 [step 1 ms]



En premier, il y a l'identifiant de l'objet de communication qui sera transmis.

Puis, il y a le type de transmission qui définit comment est transmise la donnée. Cette valeur peut aller de 0 à 255. Je reviendrai spécifiquement sur cet objet.

Le paramètre suivant est la durée minimale entre 2 messages (inhibit time) pour les PDOs en émission.

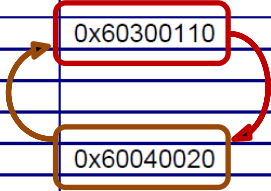
Le dernier paramètre permet de transmettre le PDO toutes les x ms avec x la valeur de l'objet sur 2 octets. Cet EventTimer est commandé par une horloge. La plage de valeurs de cette horloge s'étend donc de 1ms ... 65535 ms. Il n'est possible d'utiliser cette fonctionnalité que si le type de transmission est égal à 254 ou 255.

Tous ces paramètres sont modifiables avec des SDO.

Le PDO Mapping Parameter

L'objet Mappage 1A00h décrit le 1er PDO d'émission. Il est possible de mapper des objets jusqu'à une longueur de données maximale de 8 octets. De la même manière, l'objet 1A01h décrit le PDO d'émission 2 et l'objet 1A02h décrit le PDO d'émission 3.

1A00h	TPDO1 Mapping	
01h	1.Mapped Object	0x60040020
1A01h	TPDO2 Mapping	
01h	1.Mapped Object	0x60300110
1A02h	TPDO3 Mapping	
01h	1.Mapped Object	0x60040020



On définit le nombre d'objets mappés au sous-index 00h. Il est possible par exemple d'avoir 8 objets mappés de taille 1 octet, ou encore 2 objets de 4 octets, 1 objet de 4 octets et 2 objets de 2 octets, Un TPDO mapping intéressant est de mapper la position (4 octets) et la vitesse (2 octets).

Tous les objets du dictionnaire ne peuvent pas être mappés. Il faut donc vérifier dans celui-ci.

Mode d'émission des PDO

Comme expliqué précédemment c'est le type de transmission (valeur de 0-255) qui définit le mode. L'explication étant incomplète dans le manuel CANOpen du codeur, je vais expliquer ce que j'ai appris par expérience. Tout d'abord, quel que soit le type de transmission, l'émission du PDO peut toujours se déclencher via une trame RTR. En effet, il suffit d'envoyer une trame RTR au codeur avec l'identifiant du TPDO (ex : COB-ID TPDO1 index 1800h sous-index 01h), pour que celui émette le TPDO correspondant à l'identifiant. Voici maintenant les autres modes que j'ai pu expérimenter (de 241-251 les valeurs sont réservées) :

- Une valeur entre 1 et 240 indique une émission synchrone et cyclique du PDO. Le numéro du type de transmission indique le nombre d'impulsions SYNC nécessaire à l'émission des PDO. Pour envoyer une SYNC, il suffit simplement un identifiant égal à **080h** et l'impulsion sera générée quel que soit la donnée.
- Pour 254 et 255 il y a l'EventTimer que j'ai expliqué précédemment.
- Pour 254, le TPDO est émis à chaque changement de valeur de la donnée.

Pour les autres valeurs (0, 252 et 253), je ne peux rien dire de plus puisque seul les trames RTR me permettent d'obtenir une émission de leur part à l'heure actuelle. .

(4) Layer Setting Service (LSS)

Ce service permet de modifier efficacement le nœud du codeur ainsi que sa vitesse de transmission. Cette partie est suffisamment bien expliquée dans le manuel CANopen du codeur il n'est donc pas nécessaire que je m'étende sur le sujet. Pour les modifications de ces paramètres, il suffit d'envoyer des trames spécifiques, détaillées dans le document. Il faut par contre faire un [reset](#) après être sorti du mode LSS pour que les modifications soient effectives. Voici toutefois un lien vers un document plus complet sur le sujet :

<http://www.microcontrol.net/download/appnotes/an1204.pdf>

c) Réglages du codeur

(1) Les paramètres du bus : vitesse de transmission et numéro de nœud

Ces deux paramètres sont à régler impérativement pour chaque codeur du réseau. La vitesse de transmission est contenu dans l'objet à l'index 2100h sur 1 octet. Elle doit être la même pour tous les codeurs du réseau. Sa valeur par défaut est réglée à 250kb/s, mais la valeur de l'octet est FFh. Voici les valeurs qu'elle peut prendre :

Valeur	Vit. de trans. en kbit/s
0	10
1	20
2	50
3	100
4	125
5	250
6	500
8	1000

Le numéro de nœud, objet 2101h, est compris dans une plage de 1 à 7Fh et est sur 1 octet. Il doit être unique pour chaque codeur du réseau. Sa valeur par défaut est réglée à 3Fh, mais la valeur de l'octet est FFh (très étrange).

Ces deux paramètres sont modifiables via le service LSS. Ils peuvent également se modifier via SDO, mais il est alors nécessaire de faire une sauvegarde à l'aide de l'objet [Save all bus parameter](#) et de [reset](#) l'appareil pour que les modifications soient effectives.

(2) Save all bus parameter

Ce paramètre (objet 2105h) sauvegarde les paramètres désirés du bus (objet 2100h, 2101h) de manière permanente dans la mémoire flash. Il suffit de transmettre l'objet SDO suivant pour sauvegarder :

[Save all bus parameters](#) : 23 05 21 00 73 61 76 65

(3) Save all

Si on modifie un objet dans le dictionnaire (autres que les paramètres du bus), sa valeur ne sera effective que jusqu'au prochain **reset**. Il est donc nécessaire de faire appel à l'objet **Save all** 1010h sous-index 1h pour sauvegarder les objets. De la même manière qu'avec **Save all bus parameters**, il faut transmettre un objet SDO pour sauvegarder :

Save all: 23 10 10 01 73 61 76 65

(4) Load factory default

Pour réinitialiser tous les objets du dictionnaire à leurs valeurs standards, on doit se servir de l'objet **Restore Parameters** 1011h sous-index 1h. Il faut transmettre l'objet SDO suivant :

Load factory default : 23 11 10 01 6C 6F 61 64

Puis il faut faire un **save all bus parameters** si jamais les paramètres du bus ne sont pas aux valeurs standards. Cependant, il ne faut surtout pas faire un **save all**.

Il suffit ensuite de faire un **reset** et les objets seront tous réinitialisés.

d) Communiquer avec le périphérique IXAAT : USB to CAN

Pour apprendre à communiquer avec le codeur, un moyen simple a été d'utiliser le module USB to CAN de IXAAT. Voici le manuel d'utilisation de l'appareil :

<https://www.ixxat.com/docs/librariesprovider8/default-document-library/products/can-pc-interfaces/usb-to-can-v2-manual-english.pdf?sfvrsn=2>

A l'aide du pilote VCI et du logiciel de CANalyser que l'on installe sur l'ordinateur, on peut communiquer avec le codeur. On le trouve sur la page d'IXAAT suivante :

<https://www.ixxat.com/fr/support/telechargement-de-fichiers-et-de-documentation/pilotes/pilote-vci-v4>

Le logiciel est très pratique et très simple d'utilisation. Pour apprendre à se servir du logiciel, le Help est suffisant.

Il suffit dans un premier temps de sélectionner le périphérique et de sélectionner la bonne vitesse de transmission. Voici des exemples de trame que l'on peut envoyer, avec un affichage de l'envoi et de la réception du message :

IXXAT canAnalyser3 Mini

Controllers: 0

Receive: Overruns: 0 Errors: 0

No	Time (abs)	State	ID (hex)	DLC	Data (hex)	ASCII
150	00:15:38.496	S	0	2	81 3F	.?
151	00:15:38.508		73F	1	00	.
152	00:15:39.312	S	0	2	01 3F	.?
153	00:15:39.312		1BF	4	B4 1F 00 00
154	00:15:39.313		3BF	2	00 00	..
155	00:15:39.313		13F	6	B4 1F 00 00 B2 D0
156	00:15:39.313		140	4	4B E0 FF FF	K...
157	00:15:40.239	S	1BF	0	Remote Frame (DLC=0)
158	00:15:40.240		1BF	4	B4 1F 00 00
159	00:15:41.191	S	63F	4	40 00 18 02	@...
160	00:15:41.192		5BF	8	4F 00 18 02 FF 00 00 00	O.....
161	00:15:42.415	S	63F	6	2B 00 18 05 E8 03	+.....
162	00:15:42.416		5BF	8	60 00 18 05 00 00 00 00	*.....
163	00:15:42.416		1BF	4	B4 1F 00 00
164	00:15:43.416		1BF	4	B4 1F 00 00
165	00:15:44.416		1BF	4	B4 1F 00 00
166	00:15:45.415		1BF	4	B4 1F 00 00
167	00:15:46.415		1BF	4	B4 1F 00 00

Event timer d'un pas de 1 s

Transmit

Tx	ID (hex)	Description	Ext.	RTR	Data (hex)	Count	Time (ms)	Cycle options	Inc Mode	Byte
	0	reset node	<input type="checkbox"/>	<input type="checkbox"/>	81 3F	0	10.00		None	
	0	swith mode operationnal	<input type="checkbox"/>	<input type="checkbox"/>	01 3F	0	10.00		None	
	1BF	RTR for ask TPD01	<input type="checkbox"/>	<input checked="" type="checkbox"/>	DLC = 0	0	10.00			
	63F	Read transmission type	<input type="checkbox"/>	<input type="checkbox"/>	40 00 18 02	0	10.00		None	
	63F	Event timer TPD01	<input type="checkbox"/>	<input type="checkbox"/>	2B 00 18 05 E8 03	0	10.00		None	

Statistics

Receive Counter: 167

Error Counter: 0

Hardware

Controller: Philips SJA1000

Serial Number: HW310148

Revision: 1.6

Driver Version: 4.0.54.0

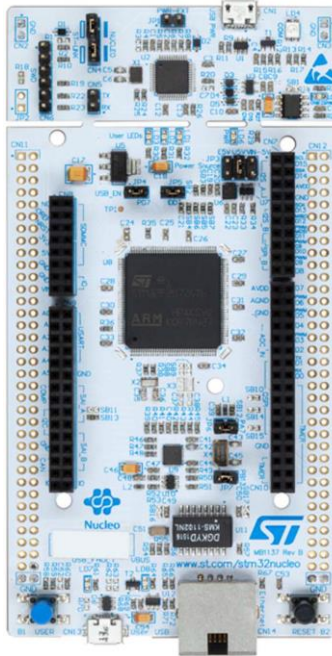
Features

- standard and extended remote-frames
- error-frames reception
- send-list
- passive mode
- delayed transmission
- single shot transmission
- high priority messages
- automatic bitrate detection
- FD Frames

III. Programmation sur STM32

A. Choix de la carte

La carte choisie est une nucléo STM32F429ZI. La raison du choix de cette carte est qu'elle dispose de 2 contrôleurs CAN pouvant ainsi gérer 2 réseaux CAN. D'autres cartes en possèdent également, mais c'est cette carte que Polytech avait en réserve.



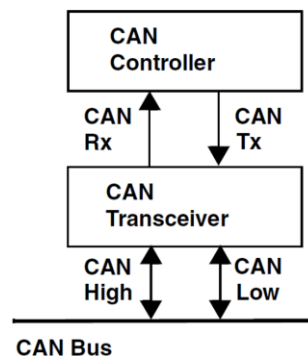
B. Contrôleur CAN

1. Tranceiver CAN

Le contrôleur CAN de la STM32 ne prend pas en charge le différentiel du CAN. Il prend en entrées CAN_RX et CAN_TX, c'est-à-dire 2 fils monodirectionnels représentant les 2 sens possibles des données sur le réseau CAN. Il faut donc un composant qui se nomme trancheiver CAN pour faire cette conversion. Nous utilisons le trancheiver SN65HVD231 de chez TI, et voici sa datasheet :

<http://www.ti.com/lit/ds/symlink/sn65hvd232.pdf>

Il est nécessaire de relier CAN_GND au trancheiver et à la carte STM32 pour avoir une masse commune. De plus, il faut alimenter le trancheiver en 3.3V que l'on peut récupérer sur un pin de la carte STM32.



2. Ce qu'il faut savoir

Tout d'abord, voici les documents nécessaires pour utiliser la STM32 :

Le User Manual :

http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/26/49/90/2e/33/0d/4a/da/DM00244518/files/DM00244518.pdf/jcr:content/translations/en.DM00244518.pdf

La Datasheet :

<http://www.st.com/content/ccc/resource/technical/document/datasheet/03/b4/b2/36/4c/72/49/29/DM00071990.pdf/files/DM00071990.pdf/jcr:content/translations/en.DM00071990.pdf>

Le Reference Manual des STM32F4 :

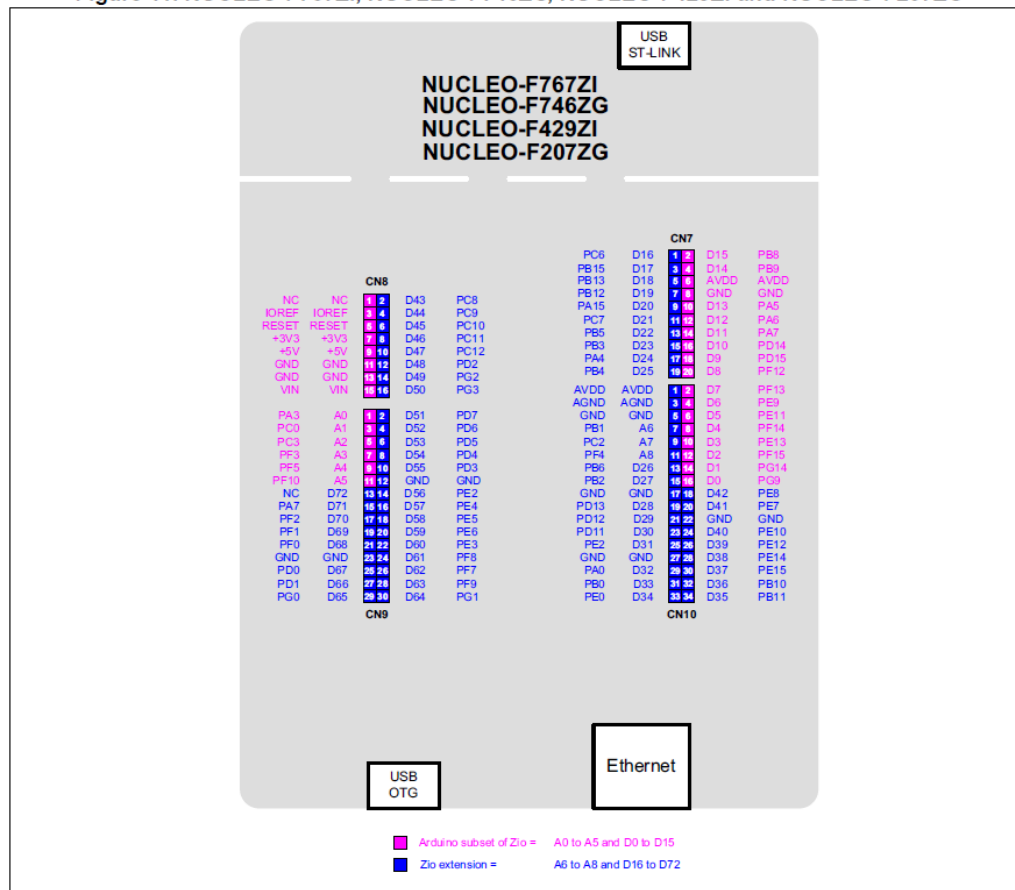
http://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

Le périphérique CAN se place au port AF9 des fonctions alternatives de la STM32. On trouve ces informations dans la Datasheet. Plusieurs entrées sont disponibles pour les entrées CAN_RX et CAN_TX des 2 contrôleurs :

CAN1 :	RX->PA11	RX->PB8	RX->PD0	RX->PI9
	TX->PA12	TX->PB9	TX->PD1	TX->PH13
CAN2 :	RX->PB5	RX->PB12		
	TX->PB6	TX->PB13		

Voici la position des pins de la carte. On trouve ce schéma dans le User Manual.

Figure 11. NUCLEO-F767ZI, NUCLEO-F746ZG, NUCLEO-F429ZI and NUCLEO-F207ZG



La documentation du contrôleur CAN se trouve dans le Reference Manual. On y trouve la liste de tous les registres utilisés par le contrôleur ainsi que des explications sur son fonctionnement. Je ne vais donc pas rentrer dans les détails.

La carte dispose de 2 contrôleurs qui supportent les versions CAN 2.0 A et B. CAN1 est le contrôleur maître et CAN2 le contrôleur esclave. Je n'ai travaillé que sur CAN1 pour le moment et je ne vais donc pas expliquer CAN2.

CAN1 dispose de 3 boîte aux lettres pour transmettre des messages (3 Tx mailboxes). Il dispose également de 2 FIFOs de réception qui peuvent chacune contenir 3 messages CAN.

En ce qui concerne la vitesse de transmission des messages, il faut bien évidemment configurer le contrôleur à la même vitesse que le réseau.

Tx mailboxes

Lors de la transmission de message, le registre de 32 bits CAN transmit status (CAN_TSR) est rempli par l'appareil. Il indique différents états que peuvent avoir les Tx mailboxes (p 1103-1105 du Reference Manual). Ce registre permet notamment de savoir quand une Tx Mailboxe est mise en attente (à cause des priorités du CAN) pour l'envoi du message.

Les registres suivants contiennent toutes les informations pour l'envoi d'un message CAN par les Tx Mailboxes (DLC, RTR, IDE, DATA,).

CAN_Ti0R	CAN_Ti1R	CAN_Ti2R
CAN_TDT0R	CAN_TDT1R	CAN_TDT2R
CAN_TDL0R	CAN_TDL1R	CAN_TDL2R
CAN_TDH0R	CAN_TDH1R	CAN_TDH2R

Three Tx Mailboxes

(p 1110-1113 du Reference Manual)

CAN_TiRx (CAN TX mailbox identifier register) est le registre qui contient l'identifiant CAN (standard ou étendu), le bit IDE, le bit RTR et le bit TXRQ qu'il faut mettre à 1 pour envoyer le message.

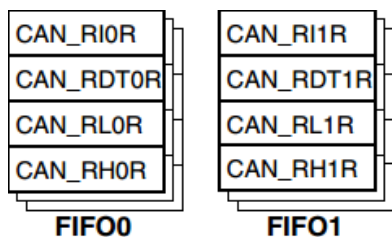
CAN_TDTxR (CAN mailbox data length control and time stamp register) contient notamment la longueur du message.

CAN_TDLxR (CAN mailbox data low register) et CAN_TDHxR (CAN mailbox data high register) sont les registres qui contiennent les 8 octets de données du message CAN.

FIFO 1 & 2

Les registres CAN_RF1R & CAN_RF2R (CAN receive FIFO x register, p 1105-1106 du Reference Manual) permettent de connaître le nombre de messages dans la FIFO (bits FMPx[1:0]), si elle est pleine (bit FULLx) et s'il y a dépassement (bit FOVRx). En mettant à 1 le bit RFOMx, on supprime le premier message de la FIFO (il est remis à 0 sitôt après par l'appareil).

Les registres suivants sont identiques à ceux des Tx_mailboxes. Ils sont remplis automatiquement à la réception d'un message sur la FIFO 1 ou 2. Ils contiennent les informations du premier message de la FIFO 1 ou 2.



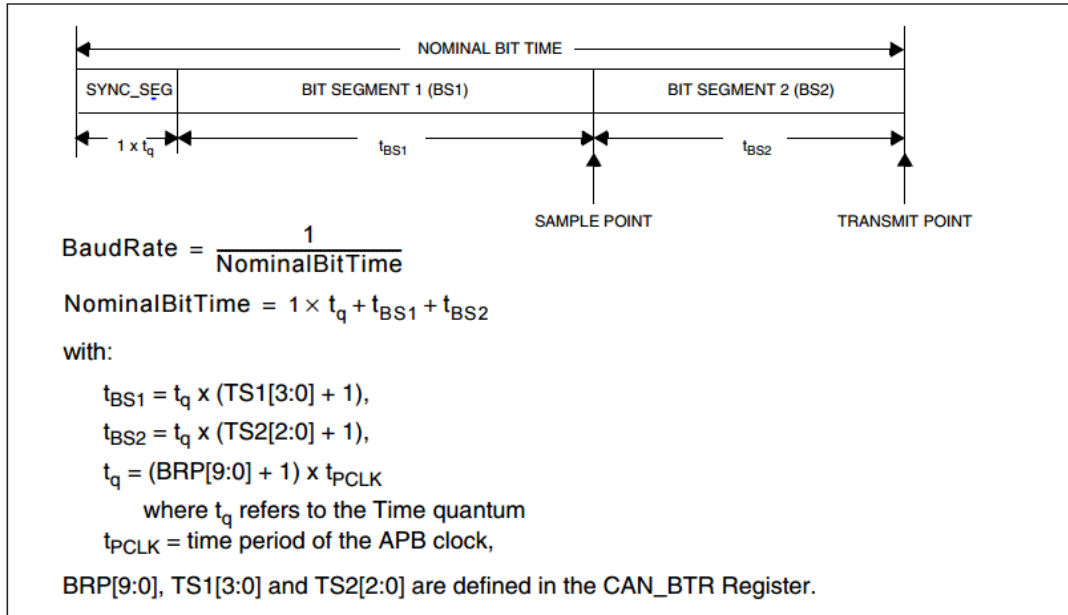
FIFO0

FIFO1

(p 1114-1116 du Reference Manual)

Vitesse de transmission

Pour configurer la vitesse de transmission il faut remplir correctement le registre CAN_BTR (CAN bit timing register, p 1109-1110 du Reference Manual). Voici comment est calculé la vitesse de transmission :



(p 1096-1097 du Reference Manual)

BS1 (bit segment 1) dure de 1 à 16 time quanta (t_q). Son nombre de time quanta est définit par $TS1 + 1$ ($TS1$ de 0 à 15).

BS2 (bit segment 2) dure de 1 à 8 time quanta (t_q). Son nombre de time quanta est définit par $TS2 + 1$ ($TS2$ de 0 à 7).

Plusieurs stratégies sont possibles pour calculer le BaudRate. Ma stratégie est de calculer BRP (Baud rate prescaler) en fixant les autres paramètres :

$$\text{BaudRate} = \frac{1}{t_q * (1 + (TS1 + 1) + (TS2 + 1))} \Leftrightarrow$$

$$t_q = \frac{1}{\text{BaudRate} * (1 + (TS1 + 1) + (TS2 + 1))} = (BRP + 1) * t_{pclk}$$

$$\text{donc } (BRP + 1) = \frac{1}{t_{pclk} * \text{BaudRate} * (1 + (TS1 + 1) + (TS2 + 1))}$$

Voici les résultats de calculs que j'ai obtenu sur excel avec une horloge APB = 42 MHz :

	Baudrate (kb/s)							
(TS1+1)+(TS2+1)	10	20	50	100	125	250	500	1000
1	2100	1050	420	210	168	84	42	21
2	1400	700	280	140	112	56	28	14
3	1050	525	210	105	84	42	21	10,5
4	840	420	168	84	67,2	33,6	16,8	8,4
5	700	350	140	70	56	28	14	7
6	600	300	120	60	48	24	12	6
7	525	262,5	105	52,5	42	21	10,5	5,25
8	466,666667	233,333333	93,333333	46,666667	37,333333	18,666667	9,333333	4,666667
9	420	210	84	42	33,6	16,8	8,4	4,2
10	381,818182	190,909091	76,363636	38,181818	30,545454	15,272727	7,636364	3,818182
11	350	175	70	35	28	14	7	3,5
12	323,076923	161,538462	64,615384	32,307692	25,846153	12,923077	6,461538	3,230769
13	300	150	60	30	24	12	6	3
14	280	140	56	28	22,4	11,2	5,6	2,8
15	262,5	131,25	52,5	26,25	21	10,5	5,25	2,625
16	247,058824	123,529412	49,411764	24,705882	19,764706	9,882353	4,941176	2,470588
17	233,333333	116,666667	46,666667	23,333333	18,666667	9,333333	4,666667	2,333333
18	221,052632	110,526316	44,210526	22,105263	17,684211	8,842105	4,421053	2,210526
19	210	105	42	21	16,8	8,4	4,2	2,1
20	200	100	40	20	16	8	4	2
21	190,909091	95,454545	38,181818	19,090909	15,272727	7,636364	3,818182	1,909091
22	182,608696	91,304348	36,521739	18,260870	14,608696	7,304348	3,652174	1,826087
23	175	87,5	35	17,5	14	7	3,5	1,75
24	168	84	33,6	16,8	13,44	6,72	3,36	1,68

Donc, pour une horloge de 42 MHz, on obtient des valeurs entières pour BRP en imposant $(TS1+1)+(TS2+1)=13$. On a donc BRP en fonction des valeurs de BaudRate que l'on désire, ce qui suffit pour configurer correctement le registre.

C. Description du programme

Le programme a été réalisé à partir du logiciel **AC6 System Workbench** qui peut se télécharger sur ce site :

<http://www.openstm32.org/HomePage>

Je ne vais décrire que la structure des fichiers principaux du programme.

Pour commencer, il y a les fichiers `can_fct.h` et `can_fct.c` qui contiennent les fonctions CAN. On a la fonction de configuration du contrôleur et de la vitesse de transmission. Il y a également les fonctions pour envoyer et recevoir des messages CAN.

Puis, il y a les fichiers `encoder_fct.h` et `encoder_fct.c` qui contiennent les fonctions CANopen pour communiquer avec le codeur.

Pour le moment, je peux faire fonctionner un codeur sur le contrôleur CAN1. Dans la suite du projet, il faudra donc essayer de faire fonctionner 2 codeurs par contrôleur (CAN1 et CAN2).

Le code est assez bien commenté. Je ne vais donc pas le réexpliquer.

IV. Ressources

A. Avant-propos

Cette partie est consacrée à énoncer les nombreuses ressources utilisées pour comprendre et utiliser le codeur. Je me suis efforcé de mettre les liens internet des fichiers en plus de les mettre dans le dossier.

Les documents sont regroupés dans un dossier que j'ai partagé sur internet au lien suivant :

<https://www.dropbox.com/sh/8ymi8f68mttg0h9/AACZMjOmG5xBN0v9pwmU0FAva?dl=0>

B. Documents pour le codeur

Les documents pour le codeur sont regroupés sur la page de Kuebler :

https://www.kuebler.com/k2014/j/fr/produkte/details/drehgeber/Absolut_Singletur n/Welle//Kompakt/M3658_CANopen

Les fichiers sont dans le dossier « Documents_Codeur » :

- **M3658-M3678_CANopen_fr.pdf** est le fichier de la datasheet du codeur. Voici le lien : https://www.kuebler.com/k2016/pdf?M3658-M3678_CANopen_fr.pdf
- **M3658-M3678_CANopen_fr.pdf** est le fichier qui contient des exemples de trame CANopen et qui explique les différents types d'objets de communication. Voici le lien : https://www.kuebler.com/k2016/pdf?R67039.0002_1_HB_AppNote_CANopen_EN_Kuebler.pdf
- **manuel_HB_Sendix_M3658_M3678_CANopen_fr_Kuebler.pdf** est le fichier de référence du codeur en ce qui concerne le CANopen. Voici le lien : https://www.kuebler.com/k2016/pdf?R67035.0003_1_HB_Sendix_M3658_M3678_CANopen_fr_Kuebler.pdf

C. Références pour le CAN

Je vais lister quelques cours sur le réseau CAN qui m'ont aidé à comprendre et à rédiger ce document. Les fichiers sont dans le dossier « Réseau_CAN » :

- **Specif_CAN.pdf** est le fichier qui contient les spécifications du CAN 2.0 écrit par BOSH. Voici le lien : <http://esd.cs.ucr.edu/webres/can20.pdf>
- Les fichiers et les dossiers qui commencent par **494-** sont des documents provenant du site eduscol.education.fr. Ils contiennent des cours et des ressources sur le réseau CAN. Voici le lien de la page des documents : http://eduscol.education.fr/sti/ressources_pedagogiques/principe-du-bus-can-et-mise-en-oeuvre
- **Part5_AutomobileCAN.pdf** est le cours sur le CAN de Mme Godary.
- **presentation_bus_CAN.pdf** est un autre cours sur le réseau CAN. Voici le lien : https://homepages.laas.fr/fcaignet/Cours/presentation_bus_CAN.pdf

D. Références pour le CANopen

Les documents expliquant le CANopen sont dans le dossiers « CANopen » :

- Le dossier **CIA** contient 2 documents techniques sur le CANopen. Il faut aller sur le site de l'organisation CIA pour les récupérer : <https://www.can-cia.org/standardization/specifications/>
- **CAN_Communication_Manual300_3-0-3.pdf** est un document très complet sur le CANopen comportant beaucoup d'exemples de trames. Voici son lien : http://www.a-m-c.com/download/sw/dw300_3-0-3/CAN_Manual300_3-0-3.pdf
- **CANopen_cours_free.ppt** est un diaporama expliquant le CAN puis le CANopen. Voici le lien : https://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwi24P-8o5DSAhWJfRoKHZfwCgAQFggjMAE&url=http%3A%2F%2Foverpof.free.fr%2Fschneider%2FCAN%2520%26%2520CANopen%2FCANopen%2FPr%25E9sentations%2FIntroduction%2FCANopen.ppt&usq=AFQjCNF7CezUFcoyJIAwK_FU02zRSvBZ5A&sig2=GXehI5QuTtgZ6RNBQYThrg
- **LSS_Configuration_Of_CANopen_Devices.pdf** est un cours sur le protocole LSS avec beaucoup d'exemples. Voici le lien : <http://www.microcontrol.net/download/appnotes/an1204.pdf>

E. Documents pour travailler sur STM32

Ces documents sont dans le dossier « **Carte_NUCLEO_STM32F29ZI** » :

- **user_manual_STM32F429.pdf** est le User Manual. Voici le lien : http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/26/49/90/2e/33/0d/4a/da/DM00244518/files/DM00244518.pdf/jcr:content/translations/en.DM00244518.pdf
- **datasheet_STM32F429.pdf** est la Datasheet. Voici le lien : <http://www.st.com/content/ccc/resource/technical/document/datasheet/03/b4/b2/36/4c/72/49/29/DM00071990.pdf/files/DM00071990.pdf/jcr:content/translations/en.DM00071990.pdf>
- **Reference_Manual_STM32F4.pdf** est le Reference Manual des STM32F4. Voici le lien : http://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf
- **can_tranceiver.pdf** est le document technique du tranceiver utilisé. Voici le lien : <http://www.ti.com/lit/ds/symlink/sn65hvd232.pdf>
- **Calcul_Baudrate_CAN.xlsx** est la feuille excel des calculs pour le baudrate.
- Le dossier « **exemple_CAN_STM** » sont des exemples de codes trouvés sur internet.
- Le dossier « **STM32F4xx_DSP_StdPeriph_Lib_V1.4.0** » est le dossier contenant les bibliothèques et les exemples de codes pour STM32F4.
- Le dossier « **encoder_project_Vf** » contient mon programme STM32.