Filière : Energie Electrique et Industrie Numérique(2-EEIN)

Activité pratique d'élément de module M34-1 :

Gestion de parking sous terrain via bus CAN

Encadré par :

- **Pr. Mr. NAITALI**

Réalisé par :

- TERRASSI Ayoub
- KADDOURI Nassira
- EL KAMOUSS Yahya
- ECHAFFAOUI Achraf

Sommaire

Liste de Figure

Liste de Tableau

Introduction

Le projet de gestion de parking souterrain est une initiative visant à instaurer un système efficace de contrôle et de gestion des places de stationnement dans un environnement souterrain. Ce type de programme est devenu de plus en plus répandu dans les zones urbaines densément peuplées, afin d'optimiser l'utilisation de l'espace de stationnement disponible. Pour garantir le bon fonctionnement du système, la mise en place d'un réseau CAN (Controller Area Network) s'avère essentielle, permettant une communication fiable et rapide entre les différents composants du système.

Dans le projet de gestion de parking souterrain, la communication par bus CAN est utilisée pour établir un protocole efficace de transmission des données entre les différents composants du système. Le protocole de communication par bus CAN repose sur le principe de diffusion générale, ce qui signifie qu'aucune station n'est spécifiquement adressée lors de la transmission d'un message. Au lieu de cela, chaque message est identifié de manière unique par tous les nœuds du réseau.

L'identification de chaque message se fait à l'aide d'un identificateur spécifique, qui est reçu par tous les nœuds du réseau. Ces nœuds sont constamment à l'écoute du réseau, ce qui leur permet de reconnaître et de traiter les messages qui leur sont destinés, en se basant sur cet identificateur.

Dans le contexte de la gestion de parking souterrain, cette approche de diffusion générale permet une communication efficace entre les différents composants du système. Par exemple, les capteurs de détection des véhicules peuvent envoyer des messages contenant des informations sur l'occupation des places de stationnement, en utilisant un identificateur spécifique. Le système central de gestion du parking, ainsi que d'autres composants tels que les panneaux d'affichage des places disponibles, les barrières d'accès, peuvent alors reconnaître ces messages et prendre les actions appropriées en fonction des informations reçues.

I-Cahier de charge :

Le projet de gestion de parking via un réseau CAN implique un système organisé autour d'un microcontrôleur en tant que nœud central, connecté à neuf nœuds TOR (Tout Ou Rien) dédiés aux capteurs et aux lampes. Le microcontrôleur, caractérisé par des spécifications techniques robustes, coordonne les opérations du système, facilitant la détection de places de parking disponibles, l'éclairage intelligent et la collecte de données. La communication CAN est utilisée pour assurer une transmission efficace entre les composants, avec des protocoles spécifiques et des fréquences définies. Les capteurs et les lampes, en tant que dispositifs TOR, sont configurés pour fonctionner selon un principe binaire. Les capteurs signalent la disponibilité des places de parking de manière tout ou rien, tandis que les lampes s'allument ou s'éteignent de manière similaire.

I-1 Protocol CAN :

Le protocole CAN (Control Area Network) est un protocole de communication série qui supporte des systèmes temps réel avec un haut niveau de fiabilité. Ses domaines d'application s'étendent des réseaux moyens débits aux réseaux de multiplexages faibles coûts. Il est avant tout à classer dans la catégorie des réseaux de terrain utilisé dans l'industrie pour remplacer la boucle analogique 20mA.

Le protocole CAN ne couvre seulement que deux des sept couches du modèle d'interconnexion des systèmes ouverts OSI de l'ISO, qui sont la couche physique et la couche liaison. La couche application est spécifiée par l'utilisateur.

I-1-1 Réseau CAN d'application

Le réseau CAN est défini comme ci-dessous :

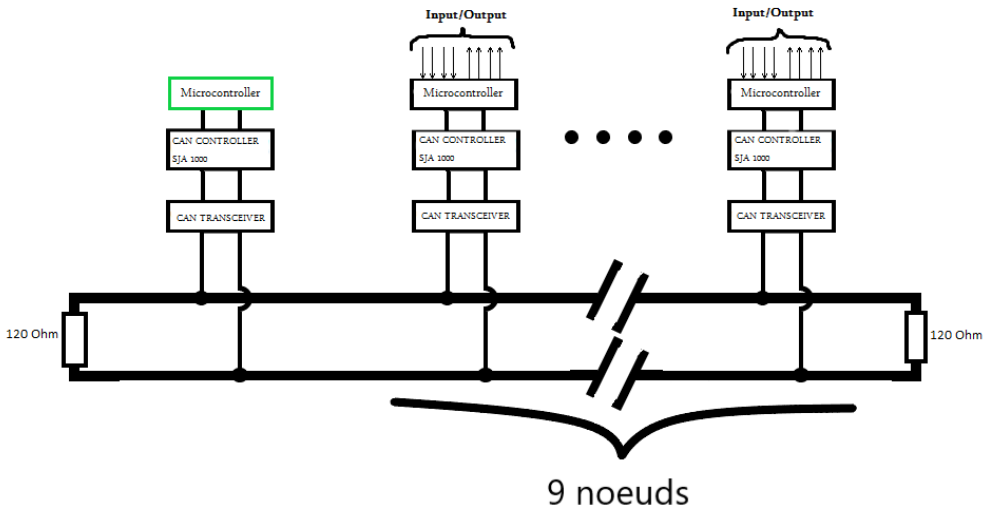


Figure 1 : Réseau CAN du projet gestion de parking

Le réseau contient :

Le CAN Controller : Il est responsable de la gestion de la communication sur le bus CAN. Il contrôle l'envoi et la réception de messages, le contrôle d'accès au médium, la gestion des erreurs, et il peut également gérer la configuration du réseau et dans notre cas on va utiliser le SJA 1000

Le CAN transceiver : Il agit comme une interface physique entre le CAN Controller et le bus physique CAN. Il gère la conversion des signaux électriques du CAN Controller en signaux appropriés.

L'impédance 120 Ohm : Il est crucial pour assurer la qualité de la communication. Elle est utilisée pour la terminaison du bus, ce qui contribue à minimiser les réflexions d'ondes et à stabiliser le signal sur le bus CAN.

Microcontrôleur : pour cette partie on va l'utiliser dans tous les nœuds, d'où le premier nœud joue le rôle de traitement et de calculateur et qu'il contient de notre programme pour répondre au cahier de charge. Et pour les autres microcontrôleurs liés au les entrées et les sorties permet de coordonner la communication, gère l'envoi et la réception de messages, encapsule les données des entrées et sorties dans des trames CAN.

I-1-2 Trames du bus CAN:

On peut diviser les trames qu'on utilise dans ce protocole selon quatre types :

La trame de donnée : elle est générée par un nœud dit "producteur" qui joue le rôle de transfert des données ou des informations, et peut être une réponse de la requête d'un autre nœud et il contient des bits suivants :

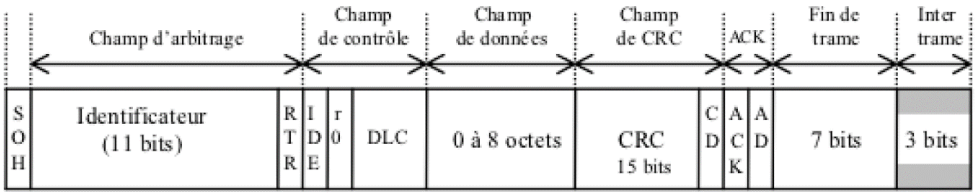


Figure 2 : constitution de la trame de donnée

La trame de requête : elle est générée par un nœud dit "consommateur" qui joue le rôle de demander des informations de la part d'un autre nœud et qu'il contient des bits suivants :

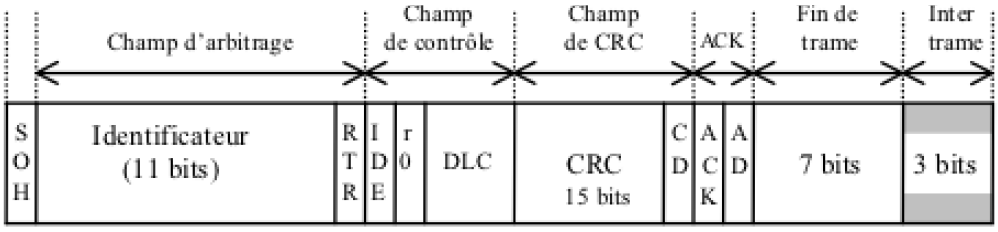


Figure 3: constitution de la trame de requête

La trame d'erreur : Permet de générer une entité du bus en réponse à la détection d'une erreur dans une trame existante.

La trame de surcharge : Elle sert à solliciter une extension du délai entre les trames de données ou les trames distantes lorsqu'elles se succèdent.

I-1-3 CAN transceiver :

La transmission des données est effectuée sur une paire filaire différentielle. La ligne est donc constituée de deux fils :

CANL (CAN LOW).

CANH (CAN HIGH).

Le CAN est un bus de terrain, soumis à des parasites importants. La transmission en paire différentielle permet de s'affranchir de ces problèmes. Les montages différentiels ont en plus un fort taux de réjection en mode commun CMRR.

Et dans cas on peut utiliser une paire torsadée la plus répondue dans une application du bus CAN et qu'elle est liée avec un transceiver CAN qui gère les niveaux électriques

La paire torsadée constitue de deux files l'une joue le rôle de CAN H et l'autre joue le rôle de CAN L qui permet de générer l'information selon le débit comme ci-dessous :

Et dans cas on peut utiliser une paire torsadée la plus répondue dans une application du bus CAN et qu'elle est liée avec un transceiver CAN qui gère les niveaux électriques

La paire torsadée constitue de deux files l'une joue le rôle de CAN H et l'autre joue le rôle de CAN L qui permet de générer l'information selon le débit comme ci-dessous :

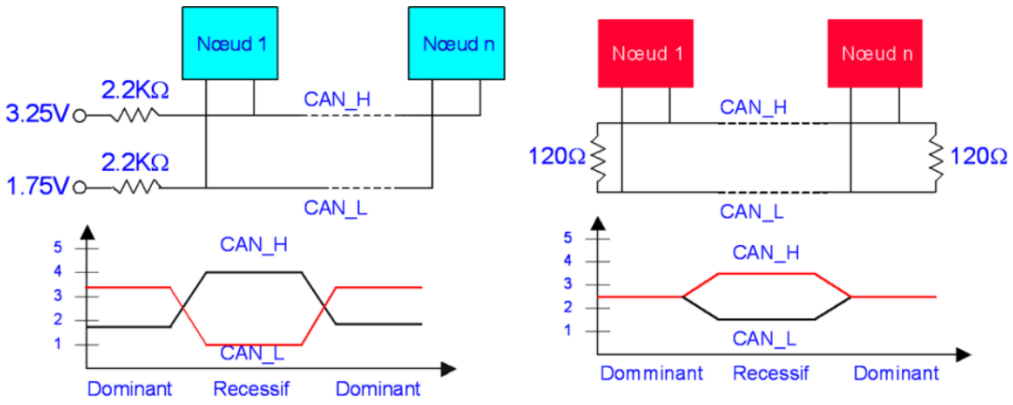


Figure 4: les types de transmission de données

Le tableau ci-dessous clarifie les principales distinctions entre les deux types de bus, en mettant en évidence spécifiquement les débits supportés :

Paramètres	CAN *low speed*	CAN *high speed*
Débit	125 kb/s	125 kb/s à 1 Mb/s
Nombre de nœuds sur le bus	2 à 20	2 à 30
Courant de sortie (mode émission)	> 1 mA sur 2,2 kΩ	25 à 50 mA sur 60Ω
Niveau dominant	CAN H = 4V CAN L = 1V	$V_{CAN\,H}$ - $V_{CAN\,L}$ = 2V
Niveau récessif	CAN H = 1,75V CAN L = 3,25V	$V_{CAN\,H}$ - $V_{CAN\,L}$ = 2,5V
Caractéristique du câble	30 pF entre les câbles de ligne	2*120Ω
Tensions d'alimentation	5V	5V

Tableau 1 :la distinction entre les deux types de bus

I-1-4 SJA 1000 :

Le SJA1000, un contrôleur CAN, est fréquemment employé dans le secteur de l'automatisation industrielle pour la mise en place du bus CAN au sein des systèmes de communication. Développé par Philips Semiconducteurs, rebaptisé NXP Semiconducteurs, ce contrôleur facilite la transmission fiable de données entre les divers nœuds du réseau.

Caractéristiques de SJA 1000 :

Réception avec un tampon de 64 octets ;

Prise en charge des standards CAN 2.0A et 2.0B ;

Débit allant jusqu'à 1 Mbit/s ;

Compteurs d'erreur avec possibilité de lecture et écriture ;

Gestion des interruptions pour chaque type d'erreur sur le bus ;

Capture des détails de la dernière erreur survenue ;

Identification du bit d'ID à l'origine d'une perte d'arbitrage ;

Mise en mode d'écoute du bus (aucune émission de signal) ;

Compatibilité avec divers microprocesseurs ;

Plage de température de fonctionnement de -40 à +125 °C.

Diagramme de SJA 1000 :

En bref, les divers blocs illustrés par la figure ci-dessous du SJA1000 exécutent les fonctions suivantes :

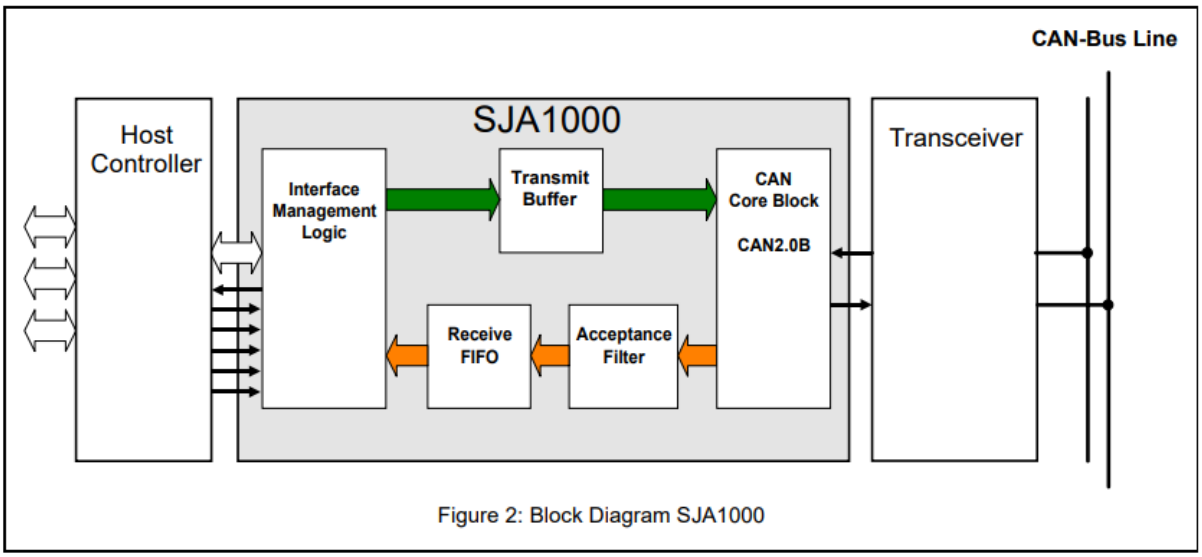


Figure 5: diagramme de SJA 1000

Il constitue :

Le bloc de base CAN (CAN Core Block) contrôle la transmission et la réception des trames CAN conformément aux spécifications CAN.

Le bloc de logique de gestion de l'interface (Interface management logic) établit une liaison avec le contrôleur hôte externe, pouvant être un microcontrôleur ou tout autre dispositif. Chaque accès au registre via le bus d'adresse/données multiplexé du SJA1000, ainsi que le contrôle des signaux de lecture/écriture, est géré dans cette unité. En plus des fonctions BasicCAN connues du PCA82C200, de nouvelles fonctionnalités PeliCAN ont été ajoutées. En conséquence, des registres et une logique supplémentaire ont été principalement mis en œuvre dans ce bloc.

Le tampon de transmission (Transmit buffer) permet de stocker un message complet (étendu ou standard). Chaque fois qu'une transmission est initiée par le contrôleur hôte, la logique de gestion de l'interface contraint le bloc de base CAN à lire le message CAN depuis le tampon de transmission.

I-2 La Supervision :

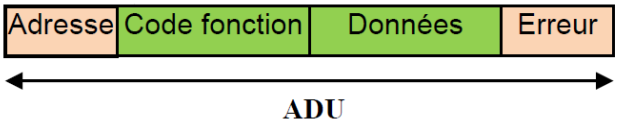
Elle s'agit du processus qui permet de surveiller, d'observer et de contrôler les différents éléments d'une installation industriel, etc…. Afin d'optimiser leur fonctionnement et d'améliorer la productivité.

I-2-1 Protocole Modbus :

Modbus est un protocole de communication standard utilisé dans les systèmes de contrôle industriel pour permettre l'échange d'informations entre dispositifs électroniques. Il repose sur une architecture maître-esclave et est largement utilisé pour la supervision et le contrôle d'appareils industriels. D'où le Modbus prend en charge plusieurs modes de communication. Les deux modes principaux sont :

Mode RTU : Il fonctionne le mode maitre-esclave d'où le maitre est actif et les esclaves sont complétement passifs ça veut dire que le maitre qui doit lire et écrire dans chaque esclave.

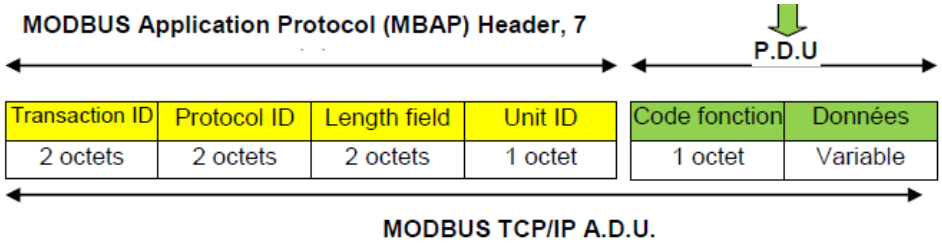
La trame Modbus RTU : On définit dans la trame MODBUS une Unité de Données de Protocole (P.D.U.), elle définit le type de fonction MODBUS et les données véhiculées et deux champs additionnels (adresse et erreur) qui constituent l'entête et le contrôle d'erreur propres à MODBUS. L'ensemble constitue l'unité de données d'application ou A.D.U.



Mode TCP : MODBUS TCP/IP utilise le protocole TCP/IP et le réseau physique ETHERNET. Il fonctionne en mode client-serveur où les clients sont actifs et les serveurs sont complétement passif ça veut dire que les clients qui doivent lire et écrire dans le serveur

Modbus et que le serveur doit identifier par un adresse IP et le numéro du port sur lequel il attend les demandes de connexion.

La trame Modbus TCP : Dans le cas de MODBUS TCP/IP, une entête spécifique notée « MBAP Header » est utilisée pour identifier l'A.D.U. MODBUS/TCP.



Utilisation dans notre cas :

Pour superviser deux parkings intelligents on utilisera le protocole de communication Modbus TCP. Ces deux parking étants des bus CAN, nous oblige à utiliser des appareils dite « passerelles » ou Gateway en Anglais. Ce dernier sera connecté à un automate qui a une interface Modbus TCP

Le schéma figuratif est le suivant :

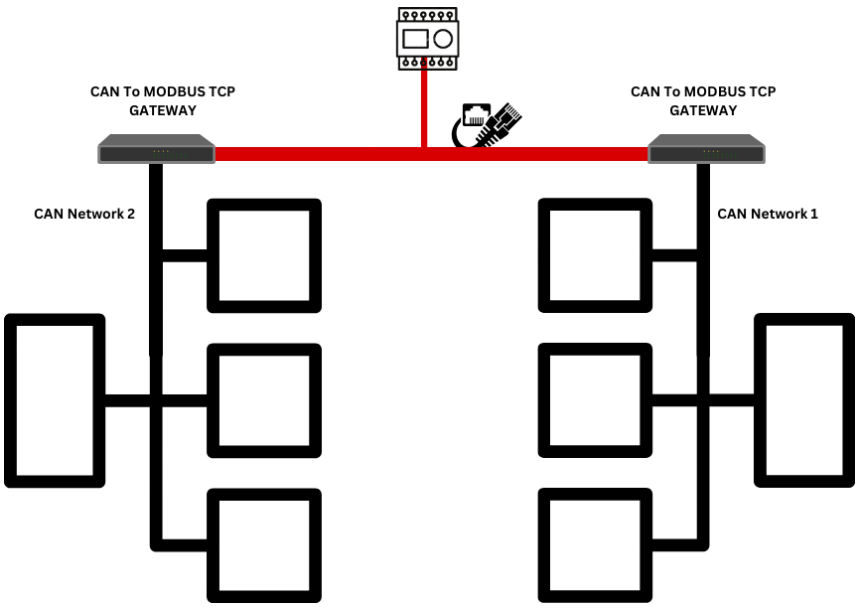


Figure 6 : supervisons de parking sous terrain avec Modbus TCP/IP

Partie II : Conception de réseaux can

II-1 Introduction

Dans le cadre de la gestion d'un parking souterrain, il est essentiel que tous les nœuds du bus CAN soient configurés avec le même débit. Le protocole CAN utilise un code NRZ (Non-Return-to-Zero) sans codage d'horloge dans la trame, ce qui nécessite une configuration précise des nœuds. Le débit nominal maximum généralement utilisé est de 1 Mbps (1 million de bits par seconde).

II-2 Calcule de time quantum

Étant donné que votre cahier des charges spécifie une longueur de bus de 50 mètres, vous pouvez directement utiliser la valeur standard de 800 kbit/s (800 000 bits par seconde) à partir d'un tableau de référence. Cette valeur est couramment utilisée pour des longueurs de bus de cette portée.

TABLE IV RECOMMENDED BIT TIMING SETTINGS					
BIT RATE (MBIT/S)	BUS LENGTH(M)	NOMINAL BIT TIME (µS)	TIME QUANTA/BIT	TIME QUANTUM (ns)	SAMPLE POINT LOCATION
1.000	25	1.0	8	125	6 TQ
0.800	50	1,25	10	125	8 TQ
0.500	100	2	16	125	14 TQ
0.250	250	4	16	250	14 TQ
0.125	500	8	16	500	14 TQ
0.050	1000	20	16	1250	14 TQ

Tableau 2:le table de recommandation de bit timing

Dans le but d'optimiser la consommation d'énergie de notre système de gestion de parking souterrain, il est nécessaire de calculer un débit plus adapté à notre configuration spécifique. Bien que le débit standard 0.8 Mbps, nous pouvons ajuster cette valeur pour répondre à nos besoins tout en assurant une communication fiable.

Une autre approche couramment utilisée pour calculer le temps de transfert (time bit) dans un réseau CAN consiste à utiliser les formules suivantes :

$$T_{bit} = \frac{1}{Debit\ nominal}$$

Cependant, il convient de noter que le temps de transfert (T_bit) peut également être déterminé en fonction du produit du nombre de bits (nb) et du temps de bit,à condition que :

$$nb * T_{bit} < Ts$$

Le nombre de bits (nb) dépend des exigences spécifiques du cahier des charges du système qui ont 10noeud 1octet pour nC et 1 octet pour nD donc :

nb=10 trames de donnée (mesure) +10 trames de données (commande) =1100bit

Avec :

10 trames de donnée (mesure) : 10(47 +8) =550

10 trames de données (commande) : 10(47 +8) =550

Donc :

$$1100{*}T_{bit} < Ts$$

$$T_{bit} < \frac{Ts}{1100} = \frac{0.05}{1100} = 45.45\mu s$$

$$Debit\ nominal = \frac{1}{T_{bit}} = \frac{1}{45.45 * 10^{-6}} = 22kbit/s$$

Cela donnera un $Debit\ nominal \approx 50kbit/s$ d'après tableau. Mais On essaye de la déterminer avec plus d'optimisation à partir de Tq :

La détermination du TQ (Time Quantum) est une étape cruciale dans la configuration du bus CAN pour notre projet de gestion de parking souterrain. Le TQ représente la plus petite unité de temps utilisée dans le système de communication CAN et est utilisé pour définir la durée de chaque bit dans la trame de données.

Pour calculer le TQ, plusieurs facteurs doivent être pris en compte, tels que :

$$\frac{Tpropa}{8} < TQ < Tpropa$$

$Tprob = 2 * 50 * 5.26 * 10^{-9} = 526ns$ Puisque le retard de câble est :5. 26ns

Donc :

$$65.75ns < TQ < 526ns$$

De même TQ peuvent être attirer de BRP qui varier entre [0…31] donc :

$$\frac{2}{F0} < TQ < \frac{64}{F0}$$

$$125ns < TQ < 4000ns$$

De sous ces contraintes on peut constater que TQ doit être :

$$T_{bitmin} = 8 * 125 = 1000ns$$

$$T_{bitmax} = 25 * 526 = 13150ns$$

Cela correspondra à un débit de 125kbit/s

Donc on peut prendre Tq=500ns donc :

$$\frac{T_{bit}}{TQ} = \frac{T_{bit}}{TQ} = \frac{8\mu s}{500ns} = 16$$

De plus On sait que TQ=$\frac{2(BRP+1)}{F0}$

Cela impliquera que $BRP = \frac{F0*TQ}{2} - 1 = 3$

II-3 Détermination des segments de bit timing

Un bit CAN est composé de quatre segments temporels contigus : le segment de synchronisation : utilisé pour la synchronisation des nœuds récepteurs sur l'émetteur, le segment de propagation, utilisé pour compenser les temps de propagations entre les différents nœuds, et les segments ″buffer phase1″ et ″buffer phase2″, utilisés pour compenser les erreurs de phase détectées lors des phases de synchronisation. La structure temporelle d'un bit est précisée sur la figure.

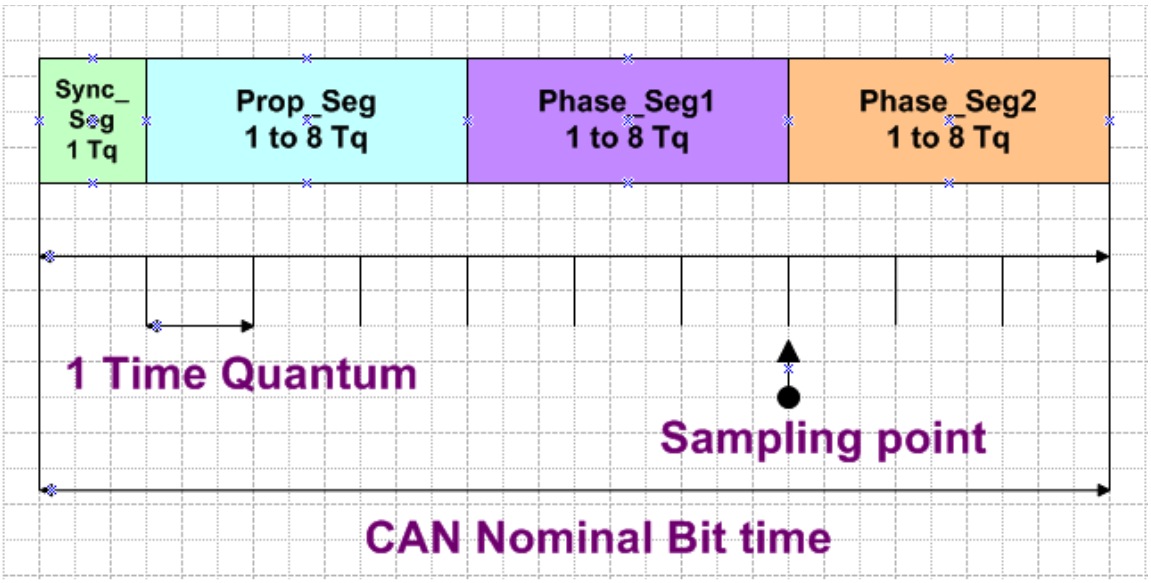


Figure 7 : segments de bit timing

TBit=TQ(SYNCSEG+PROPSEG+PHASESEG1+PHASESEG2)

Le temps de synchronisation est 1*TQ=500ns

Le temps du segment de propagation est 7*TQ=3500ns

Le temps du segment phase 1 :4*TQ=2000ns

Le temps du segment phase 2 :4*TQ=2000ns

II-4 Filtrage acceptation :

Le contrôleur CAN autonome SJA1000 est équipé d'un filtre d'acceptation polyvalent, qui permet de vérifier automatiquement l'identifiant et les octets de données. En utilisant ces méthodes de filtrage efficaces, les messages ou un groupe de messages non valides pour un nœud spécifique peuvent être empêchés d'être stockés dans le tampon de réception. Ainsi, il est possible de réduire la charge de traitement du contrôleur principal.

Le filtre est contrôlé par les registres du code d'acceptation et du masque. Les données reçues sont comparées bit par bit avec la valeur contenue dans le registre du code d'acceptation. Le registre du masque d'acceptation définit les positions des bits qui sont pertinentes pour la comparaison (0 = pertinent, 1 = non pertinent). Pour accepter un message, tous les bits reçus pertinents doivent correspondre aux bits respectifs dans le registre du code d'acceptation.

II-4-1 Filtrage d'acceptation en mode BasicCAN

Ce mode est mis en œuvre dans le SJA1000 en tant que remplacement plug-and-play (matériel et logiciel) pour le PCA82C200. Ainsi, le filtrage d'acceptation correspond aux possibilités

trouvées dans le PCA82C200 [7]. Le filtre est contrôlé par deux registres de 8 bits de largeur - le registre du code d'acceptation (ACR) et le registre du masque d'acceptation (AMR). Les 8 bits les plus significatifs de l'identifiant du message CAN sont comparés aux valeurs contenues dans ces registres. Ainsi, des groupes de huit identifiants peuvent toujours être définis pour être acceptés par n'importe quel nœud.

Example : MSB
LSB

Le registre du code d'acceptation (ACR) contient : 0 1 1 1 0 0 1 0

Le registre du masque d'acceptation (AMR) contient : 0 0 1 1 1 0 0 0

Les messages avec les identifiants suivants de 11 bits 0 1 x x x 0 1 0 x x x

sont acceptés. ID.10 ID.0

(x = ne tient pas compte)

Aux positions de bits contenant un "1" dans le registre du masque d'acceptation, n'importe quelle valeur est autorisée dans la composition de l'identifiant. La même règle s'applique aux trois bits de poids faible. Ainsi, 64 identifiants différents sont acceptés dans cet exemple. Les autres positions de bits doivent être égales aux valeurs du registre du code d'acceptation.

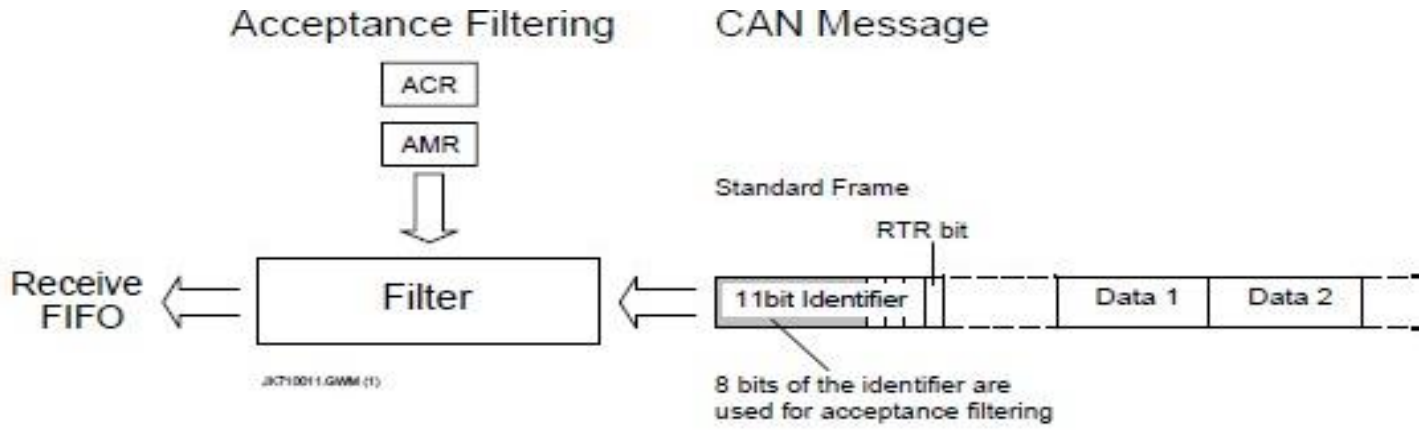


Figure 8: Filtre d'acceptance en BasicCan

Nous avons entrepris de mettre en place un système de filtrage de messages pour nos 10 nœuds, et pour ce faire, nous avons recours à des mécanismes de masquage et d'acceptation. Cette approche nous confère un contrôle précis sur les messages traités par nos 10 nœuds, garantissant ainsi que seuls les messages pertinents sont pris en considération.

Les identificateurs de trame diffusée sur le bus

	Identificateur										
	Code fonctionnel				Adresse physique						
	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	ID.2	ID.1	ID.0
PLACE 1	0	0	0	0	0	0	0	1	1	1	1
PLACE 2	0	0	0	0	0	0	1	0	1	1	1
PLACE 3	0	0	0	0	0	0	1	1	1	1	1
PLACE 4	0	0	0	0	0	1	0	0	1	1	1
PLACE 5	0	0	0	0	0	1	0	1	1	1	1
PLACE 6	0	0	0	0	0	1	1	0	1	1	1
PLACE 7	0	0	0	0	0	1	1	1	1	1	1
PLACE 8	0	0	0	0	1	0	0	0	1	1	1
LCD	0	0	0	0	1	0	0	1	1	1	1
Controller	0	0	0	0	1	0	1	0	1	1	1

Acceptance et masquage de 10 nœuds :

Nœud 1

PLACE 1	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	0	0	0	1

Nœud 2

PLACE 2	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	0	0	1	0

Nœud 3

PLACE 3	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	0	0	1	1

Nœud 4

PLACE 4	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	0	1	0	0

Nœud 5

PLACE 5	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	0	1	0	1

Nœud 6

PLACE 6	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	0	1	1	0

Nœud 7

PLACE 7	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	0	1	1	1

Nœud 8

PLACE 8	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	1	0	0	0

Nœud 9

LCD	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	0	0	0	0	1	1	1	1
Acceptance	X	X	X	X	1	0	0	1

Nœud 10

Controller	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
Mask	1	1	1	1	0	0	0	0
Acceptance	0	0	0	0	X	X	X	X

II-5 Programme SJA1000

Comme son nom l'indique, cette partie du programme consiste à définir les registres internes du SJA1000 en attribuant un nom à chaque adresse, ce qui rend le programme plus lisible.

REGISTER	BIT	SYMBOL	NAME	VALUE	
				RESET BY HARDWARE	SETTING BIT CR.0 BY SOFTWARE OR DUE TO BUS-OFF
Control	CR.7	–	reserved	0	0
	CR.6	–	reserved	X	X
	CR.5	–	reserved	1	1
	CR.4	OIE	Overrun Interrupt Enable	X	X
	CR.3	EIE	Error Interrupt Enable	X	X
	CR.2	TIE	Transmit Interrupt Enable	X	X
	CR.1	RIE	Receive Interrupt Enable	X	X
	CR.0	RR	Reset Request	1 (reset mode)	1 (reset mode)
Command	CMR.7	–	reserved	note 3	note 3
	CMR.6	–	reserved		
	CMR.5	–	reserved		
	CMR.4	GTS	Go To Sleep		
	CMR.3	CDO	Clear Data Overrun		
	CMR.2	RRB	Release Receive Buffer		
	CMR.1	AT	Abort Transmission		
	CMR.0	TR	Transmission Request		
Status	SR.7	BS	Bus Status	0 (bus-on)	X
	SR.6	ES	Error Status	0 (ok)	X
	SR.5	TS	Transmit Status	0 (idle)	0 (idle)
	SR.4	RS	Receive Status	0 (idle)	0 (idle)
	SR.3	TCS	Transmission Complete Status	1 (complete)	X
	SR.2	TBS	Transmit Buffer Status	1 (released)	1 (released)
	SR.1	DOS	Data Overrun Status	0 (absent)	0 (absent)
	SR.0	RBS	Receive Buffer Status	0 (empty)	0 (empty)
Interrupt	IR.7	–	reserved	1	1
	IR.6	–	reserved	1	1
	IR.5	–	reserved	1	1
	IR.4	WUI	Wake-Up Interrupt	0 (reset)	0 (reset)
	IR.3	DOI	Data Overrun Interrupt	0 (reset)	0 (reset)
	IR.2	EI	Error Interrupt	0 (reset)	X; note 4
	IR.1	TI	Transmit Interrupt	0 (reset)	0 (reset)
	IR.0	RI	Receive Interrupt	0 (reset)	0 (reset)

II-5-1 Configuration des registres :

```
/* definition for direct access to 8051 memory areas */
#define XBYTE ((unsigned char volatile xdata *) 0)

/* address and bit definitions for the Mode & Control Register */
#define ModeControlReg XBYTE[0]
#define RM_RR_Bit 0x01 /* reset mode (request) bit */
#define RM_NR_Bit 0xFE /* Normal mode (request) bit */

#define InterruptEnReg XBYTE[0] /* Control Register */
#define RIE_Bit 0x02 /* Receive Interrupt enable bit */
#define TIE_Bit 0x04 /* Transmit Interrupt enable bit */
#define EIE_Bit 0x08 /* Error Interrupt enable bit */
#define DOIE_Bit 0x10 /* Overrun Interrupt enable bit */

/* address and bit definitions for the Command Register */
#define CommandReg XBYTE[1]
#define TR_Bit 0x01 /* transmission request bit */
#define AT_Bit 0x02 /* abort transmission bit */
#define RRB_Bit 0x04 /* release receive buffer bit */
#define CDO_Bit 0x08 /* clear data overrun bit */

/* address and bit definitions for the Status Register */
#define StatusReg XBYTE[2]
#define RBS_Bit 0x01 /* receive buffer status bit */
#define TBS_Bit 0x04 /* transmit buffer status bit */
#define TCS_Bit 0x08 /* transmission complete status bit */


/* address and bit definitions for the Interrupt Register */
#define InterruptReg XBYTE[3]
#define RI_Bit 0x01 /* receive interrupt bit */
#define TI_Bit 0x02 /* transmit interrupt bit */

/* address and bit definitions for the Bus Timing Registers */
#define BusTiming0Reg XBYTE[6]
#define BusTiming1Reg XBYTE[7]

/* address definitions of Acceptance Code & Mask Registers */
#define AcceptCodeReg XBYTE[4]
#define AcceptMaskReg XBYTE[5]

* address definitions for Rx-Buffer */
#define RxBuffer1 XBYTE[20]
#define RxBuffer2 XBYTE[21]
#define RxBuffer3 XBYTE[22]
#define RxBuffer4 XBYTE[23]
```

```
#define RxBuffer5 XBYTE[24]
#define RxBuffer6 XBYTE[25]
#define RxBuffer7 XBYTE[26]
#define RxBuffer8 XBYTE[27]
#define RxBuffer9 XBYTE[28]
#define RxBuffer10 XBYTE[29]

/* address definitions for Tx-Buffer */
#define TxBuffer1 XBYTE[10]
#define TxBuffer2 XBYTE[11]
#define TxBuffer3 XBYTE[12]
#define TxBuffer4 XBYTE[13]
#define TxBuffer5 XBYTE[14]
#define TxBuffer6 XBYTE[15]
#define TxBuffer7 XBYTE[16]
#define TxBuffer8 XBYTE[17]
#define TxBuffer9 XBYTE[18]
#define TxBuffer10 XBYTE[19]

int data[8];  /* Define the data array as a global variable*/
int id;  /* Define the id as a global variable*/
```

II-5-2 Configuratin de SJA1000:

```
bool Config() { /*SJA1000 configuration function to be completed…*/
flag=FALSE;
/* Select the RESET mode*/
/*
ControlReg : | * | * | * |OIE_Bit|EIE_Bit|TIE_Bit|RIE_Bit|RR_Bit| ;
RM_RR_Bit=0x01;
CommandReg : | * | * | * | * |CDO_Bit|RRB_Bit|AT_Bit|TR_Bit|
StatusReg : | * | * | * | * |TCS_Bit|TBS_Bit| * |RBC_Bit|
ControlReg : | * | * | * | * | * | * |TI_Bit|RI_Bit|

*/
while ( ( ModeControlReg=RM_RR_Bit )!= RM_RR_Bit );
    /* all interupts are then disabled */
    OIE_Bit=0;
    EIE_Bit=0;
    TIE_Bit=0;
    RIE_Bit=0;

    /* configuration for acceptance code and mask registers */
    AcceptCodeReg=0x00;
    AcceptMaskReg=0xF0;

    /* configuration of bus timing */
    /* bus timing values for
    - bit-rate : 125 kBit/s
    - oscillator frequency : 16 MHz */
    /* baud rate prescaler : 3 */
    /* SJW : 1 */
    /* TSEG1 : 11 */
    /* TSEG2 : 4 */
    /* SPL:1  the bus is sampled 3 times*/

    #define btr0 0x03
    #define btr1 0xBB


/* Normal mode request */
while( ( ModeControlReg &= RM_NR_Bit) & 0x01 );
    Return flag;
    /* all interupts are renabled */
    OIE_Bit=1;
    EIE_Bit=1;
    TIE_Bit=1;
    RIE_Bit=1;
}
```

II-5-3 IRQ_CALLBACK (reception interrupt callback):

```
// Define the reception interrupt Callback Function
IRQ_CALLBACK void SJA1000_IRQCallback(InterruptEnReg) {

    if (InterruptEnReg == RI_BIT) {
    /* configuration of bus timing */
    OIE_Bit=0;
    EIE_Bit=0;
    TIE_Bit=0;
    RIE_Bit=0;


        // Handle received messages tasks
    if (StatusReg == RBS_Bit) { /* Check if receive status = full*/
            // Stock received data frame
            id = (RxBuffer1 << 3) | ((RxBuffer2 & 0xE0) >> 5); /* Data frame
id 11 bits*/
            data[0] = RxBuffer3; /The first byte/
            data[1] = RxBuffer4;
            data[2] = RxBuffer5;
            data[3] = RxBuffer6;
            data[4] = RxBuffer7;
            data[5] = RxBuffer8;
            data[6] = RxBuffer9;
            data[7] = RxBuffer10;

           CommandReg=RRB_Bit; /Release receive buffer/
        }
    }
/* all interupts are renabled */
    OIE_Bit=1;
    EIE_Bit=1;
    TIE_Bit=1;
    RIE_Bit=1;
}
```

II-5-4 Can_transmit(Transmission Function):

```
/* transmission function definition*/
void can_transmit(int id1, int id2, int RTR, int data)

{

    /* all interupts are then disabled */
    OIE_Bit=0;
    EIE_Bit=0;
    TIE_Bit=0;
    RIE_Bit=0;

    if(StatusReg == TBS_Bit) /* or test transsmit status bit */
        {
            if (RTR==0) /* transsmit data frame */
            {

                TxBuffer1 = id1;
                TxBuffer2 = id2;
                TxBuffer3 = data[0]; /* the first byte */
                TxBuffer4 = data[1];
                TxBuffer5 = data[2];
                TxBuffer6 = data[3];
                TxBuffer7 = data[4];
                TxBuffer8 = data[5];
                TxBuffer9 = data[6];
                TxBuffer10 = data[7];
            }

            else
                TxBuffer1 = id1;
                TxBuffer2 = id2;

        }
/* Start the transmission */
TR_Bit== 1 ; /* Set Transmission Request bit */
if(StatusReg == TCS_Bit) /* transsmission complete */
StatusReg = 0x00; /* disable flags transmission */
/* all interupts are renabled */
    OIE_Bit=1;
    EIE_Bit=1;
    TIE_Bit=1;
    RIE_Bit=1;
}
```

II-5-5 Main :

```
void main() {

   /* Initialize CAN */
     if (Config()) {
        while (1) {

           /* Check if a message is received */
           if (StatusReg & RBS_Bit) {
               // Call reception callback function
               SJA1000_IRQCallback(InterruptEnReg);

             /* if data = 0x0F then the sensor did not detect the presence of
a car,
         so i will send a command of 0x0F which will be interpreted bt the
         other node with green LED ON and red LED OFF.
         And if de data i recieve is 0xF0 the the ditetcor detected a car
         so i will send data =0xF0 which means green LED must be OFF and red
LED must be ON */

         can_transmit(YOUR_TRANSMIT_ID1, YOUR_TRANSMIT_ID2, 0, data);


               /* Clear the flag*/
               StatusReg &= ~RBS_Bit;
           }


        }
    }
}
```

Pour la configuration du contrôleur SJA1000 des autres nœuds ça sera la même chose, seulement les filtres qui diffèrent (les deux registres AcceptCodeReg et AcceptMaskReg qui changent selon le filtre de chaque nœud). Leur fonction main aussi sera différente, dans un premier temps il y aura une appelé de la fonction d'initialisation Config(), puis le traitement suivant en continu: si il y a une interruption sur RX, il fait appelle a SJA1000_IRQCallback() et lire la commande reçu, si c set 0x0F alors il vas allumer la LED verte et éteindre la LED rouge et si il s'agir de 0xF0 donc il va allumer la LED rouge et éteindre la LED verte.

Conclusion

Autour de cette activité pratique, on a bien appris et maîtrisé la manière de réalisation et étude du protocole CAN et Modbus qui jouent un rôle très important dans l'industrie. Cette activité pratique nous à pousser à développer notre expérience vers le calcul d'ingénieur dans un projet qui dans notre cas était l'application "gestion de parking".

La première partie consiste à l'étude pratique de la réalisation, avant de commencer les calculs. On a besoin de savoir pourquoi on utilise spécifiquement le bus CAN et pas un autre bus, l'utilité du SJA 1000 dans cette application et le Modbus et son rôle de supervision dans cette application.

La deuxième partie consiste à la conception, où on commence par les calculs pour atteindre notre objectif tel que le time quantum, bit timing...etc, car cette étape est essentielle pour connaitre les éléments nécessaires pour effectuer une communication dans un bus CAN et spécifiquement pour configurer le SJA1000. Lorsqu'on a bien maîtrisé le bus CAN dans les différents coté on a mis en place l'idée principal pour un système de supervision en utilisant le Modbus TCP qui aide à effectuer la supervision sur deux copies du réseau CAN qu'on a étudié.

Ces deux étapes, nous à bien entraîner sur des élément très nécessaire dans le monde industriel.