

**MASTER AUTOMATIQUE ROBOTIQUE ET INFORMATIQUE APPLIQUEE**

**SPECIALITE : TEMPS REEL, CONDUITE ET SUPERVISION**

**Année 2017 / 2018**

**Thèse de Master**

Présenté et soutenu par :

**BOURAGBA Salah-Eddine**

Le  
**30/07/2018**

À  
**Ecole Centrale de Nantes**

**TITRE**

**Ecriture d'un pilote pour le contrôleur MCP2517FD et vérification du programme générateur de trames**

**JURY**

**Président : Jean-Luc BÉCHENNEC**

**Examinateur : Jean-Jacques LOISEAU**

**Directeur de thèse : Pierre MOLINARO**

**Laboratoire : LS2N**

# Introduction générale

Le protocole CAN est un protocole de communication série qui supporte des systèmes temps réel avec un haut niveau de fiabilité. Il est utilisé principalement dans les domaines de l'automobile et de l'aéronautique. Le CAN FD a été introduit par la suite, pour améliorer l'utilisation de la bande passante et la vitesse du CAN.

Ce projet est une continuation du sujet de séminaire bibliographique intitulé “étude de la différence entre le NON ISO CAN FD et l'ISO CAN FD, et réalisation d'un programme générateur de trames”. Le but du projet est d'écrire un pilote pour le contrôleur CAN FD **MCP2517FD** et de vérifier le programme générateur de trames.

Dans la première partie de ce rapport, nous allons commencer par la définition et les propriétés du réseau CAN, ensuite, nous allons voir en détail la composition des trames CAN 2.0B et ISO CAN FD, et enfin, nous allons montrer la différence entre ces deux types de trames.

Dans la deuxième partie du rapport, nous allons voir la configuration des *Nominal et Data bit time*. Nous allons utiliser des algorithmes basés sur les équations de la tolérance d'horloge du standard **ISO 11898-1 2015**.

Dans la troisième partie, nous allons voir la composition de la bibliothèque **MCP2517FD**, nous allons commencer par les fonctions de configuration du contrôleur, ensuite, nous allons voir les fonctions d'envoi et de réception des trames, et enfin, nous allons voir un exemple de l'utilisation de la bibliothèque en mode *InternalLoopback*.

Dans la quatrième partie du rapport, nous allons tester dans un premier lieu, la bibliothèque en mode de fonctionnement normal, et dans un deuxième lieu, nous allons tester le programme générateur de trames.

# Liste des figures

1.1	Composition des trames CAN 2.0B . . . . .	11
1.2	Composition des trames ISO CAN FD . . . . .	12
1.3	Les vitesses de transmission du CAN FD selon la charge . . . . .	13
2.1	CAN FD bit time [1] . . . . .	17
2.2	Le temps de propagation entre deux stations CAN [2] . . . . .	19
3.1	Le diagramme de contexte du MCP2517FD . . . . .	28
3.2	Le diagramme de blocs du MCP2517FD . . . . .	29
3.3	Le diagramme de classe du MCP2517FD . . . . .	29
3.4	Le constructeur par défaut de la classe MCP2517FD . . . . .	30
3.5	Les modes de configuration du contrôleur . . . . .	31
3.6	Le diagramme de la classe CANFDFrame . . . . .	35
3.7	Le schéma de couplage en mode InternalLoopback . . . . .	37
3.8	Test de la bibliothèque MCP2517FD en mode <i>InternalLoopback</i> . . . . .	40
4.1	Le schéma de couplage en mode Normal . . . . .	42
4.2	Test de la bibliothèque MCP2517FD en mode <i>Normal</i> . . . . .	45
C.1	Le diagramme de la classe embeddedvector <TYPE> . . . . .	55
D.1	Le bit banding dans le <b>Cortex M4</b> . . . . .	59

# Liste des tableaux

2.1	La taille des paramètres du bit time [2] . . . . .	17
B.1	Le temps d'exécution des opérations en entier et en flottant . . . . .	50

# Liste des algorithmes

2.1	Calcul du <i>NominalPrescaler</i> et <i>NominalBitTimeQuantum</i> . . . . .	18
2.2	Configuration du <i>NominalBitTime</i> . . . . .	21
2.3	Vérification de la tolérance d'horloge du <i>NominalBitTime</i> . . . . .	23
2.4	Calcul du <i>DataPrescaler</i> et <i>DataBitTimeQuantum</i> . . . . .	23
2.5	Configuration du <i>DataBitTime</i> . . . . .	25
2.6	Vérification de la tolérance d'horloge du <i>DataBitTime</i> . . . . .	26

# Nomenclature

BRS Bit Rate Switch

CAN Controller Area Network

CAN FD CAN with Flexible DataRate

CRC Cyclic Redundancy Check

DLC Data Length Code

EM ElectroMagnetique

EOF End Of Frame

ESI Error State Indicator

GPIO General Purpose Input/Output

IDE IDentifier Extension

ISO International Organization for Standardization

MISO Master In Slave Out

MOSI Master Out Slave In

PPM Part Per Million

RAM Random Access Memory

RTR Remote Transmission Request

SAE Society of Automotive Engineers

SCLK Serial Clock

SJW Synchronization Jump Width

SOF Start Of Frame

SPI Serial Periphiral Interface

TTCAN Time Triggered CAN

UML Unified Modeling Language

# Table des matières

<b>Introduction générale</b>	<b>1</b>
<b>Liste des figures</b>	<b>2</b>
<b>Liste des tableaux</b>	<b>3</b>
<b>Liste des algorithmes</b>	<b>4</b>
<b>Nomenclature</b>	<b>5</b>
<b>Table des matières</b>	<b>8</b>
<b>1 Introduction au CAN FD</b>	<b>9</b>
1 Historique et définitions . . . . .	10
1.1 Historique . . . . .	10
1.2 Définitions . . . . .	10
1.3 Propriétés . . . . .	11
2 Composition des trames CAN 2.0B et ISO CAN FD . . . . .	11
2.1 Composition des trames CAN 2.0B . . . . .	11
2.2 Composition des trames ISO CAN FD . . . . .	12
3 Différence entre les trames CAN2.0B et ISO CANFD [3] . . . . .	12
3.1 Taille de la trame . . . . .	12
3.2 Utilisation de la bande passante . . . . .	13
3.3 Vitesse de transmission . . . . .	13
3.4 Calcul du CRC . . . . .	14

<b>2 Configuration du bit time</b>	<b>15</b>
1 Composition du bit time . . . . .	16
1.1 Définitions [2] . . . . .	16
1.2 La composition du bit time . . . . .	17
1.3 Les valeurs des paramètres du bit time . . . . .	17
2 Configuration du bit time . . . . .	18
2.1 Configuration du nominal bit time . . . . .	18
2.2 Configuration du data bit time . . . . .	23
<b>3 La bibliothèque MCP2517FD</b>	<b>27</b>
1 Introduction au contrôleur [4] . . . . .	28
2 Introduction à la bibliothèque . . . . .	28
3 Diagramme de classe . . . . .	29
3.1 Les attributs de la classe . . . . .	30
3.2 Les constructeurs de la classe . . . . .	30
4 Les méthodes de configuration . . . . .	30
4.1 Configuration de la communication SPI . . . . .	30
4.2 Configuration du bitTime . . . . .	31
4.3 Configuration du mode d'opération . . . . .	31
4.4 Configuration du TXQ (Transmit Queue) . . . . .	32
4.5 Configuration des FIFOs . . . . .	32
4.6 Configuration TEF (Transmit Event FIFO) . . . . .	33
4.7 Configuration des filtres . . . . .	33
4.8 Configuration de l'horodatage (Time Stamping) . . . . .	34
5 Les méthodes d'envoi et de réception . . . . .	35
5.1 La classe CANFDFrame . . . . .	35
5.2 Les constructeurs de la classe . . . . .	35
5.3 Les méthodes d'envoi . . . . .	36

5.4	La méthode de réception . . . . .	36
6	Test de la bibliothèque en mode LoopBack . . . . .	37
6.1	Configuration du matériel . . . . .	37
6.2	Configuration du MCP2517FD . . . . .	38
6.3	Envoi et réception des trames . . . . .	39
<b>4</b>	<b>Test de la bibliothèque en mode normal et vérification du programme générateur de trames CAN2.0B et CANFD</b>	<b>41</b>
1	Introduction . . . . .	42
2	Configuration du matériel . . . . .	42
3	Test du programme générateur de trames et du mode normal de la bibliothèque MCP2517FD . . . . .	43
<b>Conclusion</b>		<b>46</b>
<b>Bibliographie</b>		<b>47</b>
<b>A Calcul de l'intervalle du PhaseSeg2</b>		<b>48</b>
<b>B Vérification de la portée des variables (range check)</b>		<b>50</b>
1	Configuration du <i>bitTime</i> . . . . .	50
<b>C La classe embeddedvector &lt;TYPE&gt;</b>		<b>55</b>
1	Les attributs de la classe . . . . .	55
2	Les constructeurs de la classe . . . . .	56
3	Les méthodes de la classe . . . . .	56
<b>D Le bit banding</b>		<b>58</b>
<b>E Les codes d'erreur de la configuration du bit time</b>		<b>60</b>

# Chapitre 1

## Introduction au CAN FD

Dans ce chapitre, nous allons voir une introduction au réseau CAN. Nous allons commencer par la définition et les propriétés du réseau CAN. Ensuite, nous allons voir en détail la composition des trames CAN 2.0B et ISO CAN FD. Enfin, nous allons montrer la différence entre ces deux types de trames.

# 1 Historique et définitions

## 1.1 Historique

- En 1986, Robert Bosch GmbH a introduit le CAN au congrès SAE.
- En 1987, les deux sociétés Intel et Philips Semiconductors ont lancé les deux premiers microcontrôleurs CAN.
- En 1991, Bosch a publié la version CAN 2.0.
- En 1992, les premières voitures Mercedes-Benz utilisent le CAN.
- En 1993, la standardisation du CAN en ISO 11898.
- En 1995, la sortie de la version étendue CAN 2.0B.
- En 2000, le développement du TTCAN et standardisation en 2004 en ISO 11898-4.
- En 2012, Bosch présente la version CAN FD.
- En 2015, standardisation du CAN FD en ISO 11898-1.

## 1.2 Définitions

Le CAN est un protocole de communication série qui supporte des systèmes embarqués temps réel avec un haut niveau de sécurité. Il est conçu pour fonctionner dans les milieux pollués (perturbations EM).

Le CAN fonctionne en architecture multi-maîtres et le transfert des messages se fait en « Broadcasting ».

Le CAN autorise différentes stations à accéder simultanément au bus, en utilisant un mécanisme d'arbitrage non destructif.

### 1.3 Propriétés

- Priorité entre les messages.
- Garantie du temps de latence.
- Souplesse de la configuration.
- Détection et signalisation des erreurs.
- Retransmission automatique des messages corrompus dès que le bus est à nouveau au repos.
- Distinction d'erreurs d'ordre temporaires ou de non-fonctionnalité permanente d'un nœud.
- Déconnexion automatique des nœuds défectueux.

## 2 Composition des trames CAN 2.0B et ISO CAN FD

### 2.1 Composition des trames CAN 2.0B

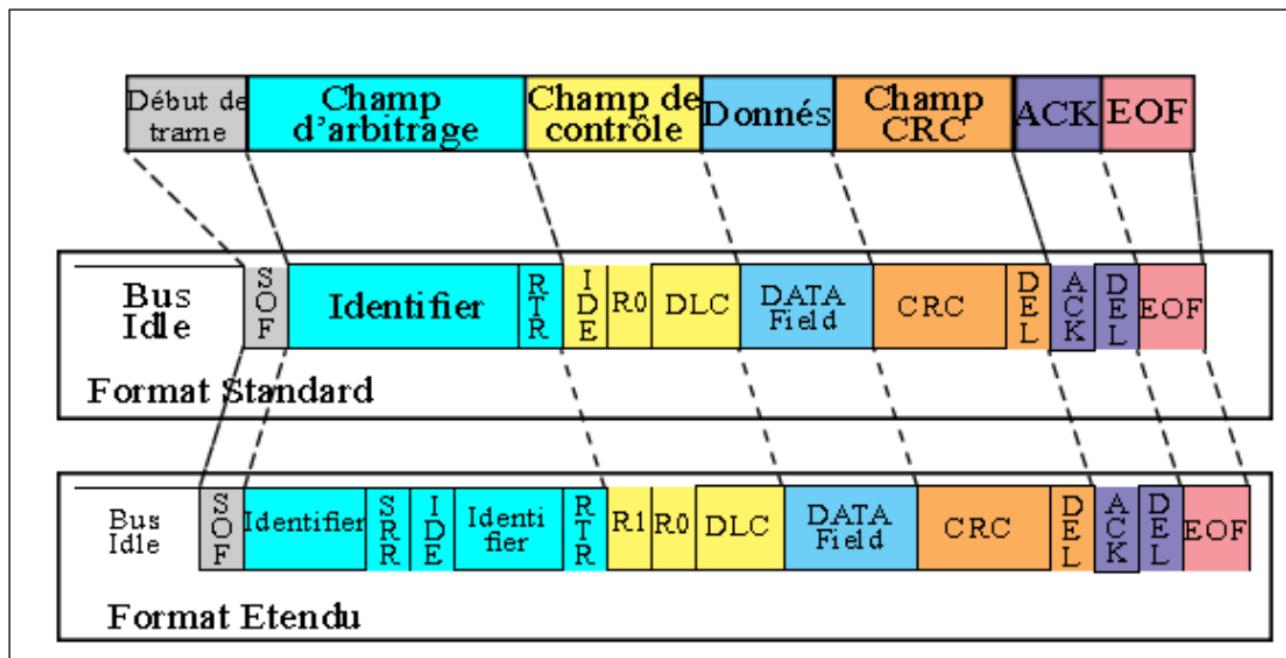


FIGURE 1.1 – Composition des trames CAN 2.0B

- **SOF** : 1 bit dominant qui détermine le début de la trame.
- **Identifiant** : 11 bits (29 bits pour les trames étendus) pour identifier chaque trame.
- **RTR** : pour différencier entre les trames de donnée (dominant) et les trames de requête (récessif).
- **IDE** : pour différencier entre les trames standards (dominant) et les trames étendues (récessif).
- **DLC** : 4 bits pour déterminer la taille du champ de donnée.
- **Champs de donnée** : contient les données à envoyer (pour les trames de requêtes, le champ est vide et  $DLC='0000'$ ).
- **CRC** : contient la séquence CRC suivie du bit récessif ‘CRC délimiteur’.
- **EOF** : 7 bits récessifs pour déterminer la fin de la trame.

## 2.2 Composition des trames ISO CAN FD

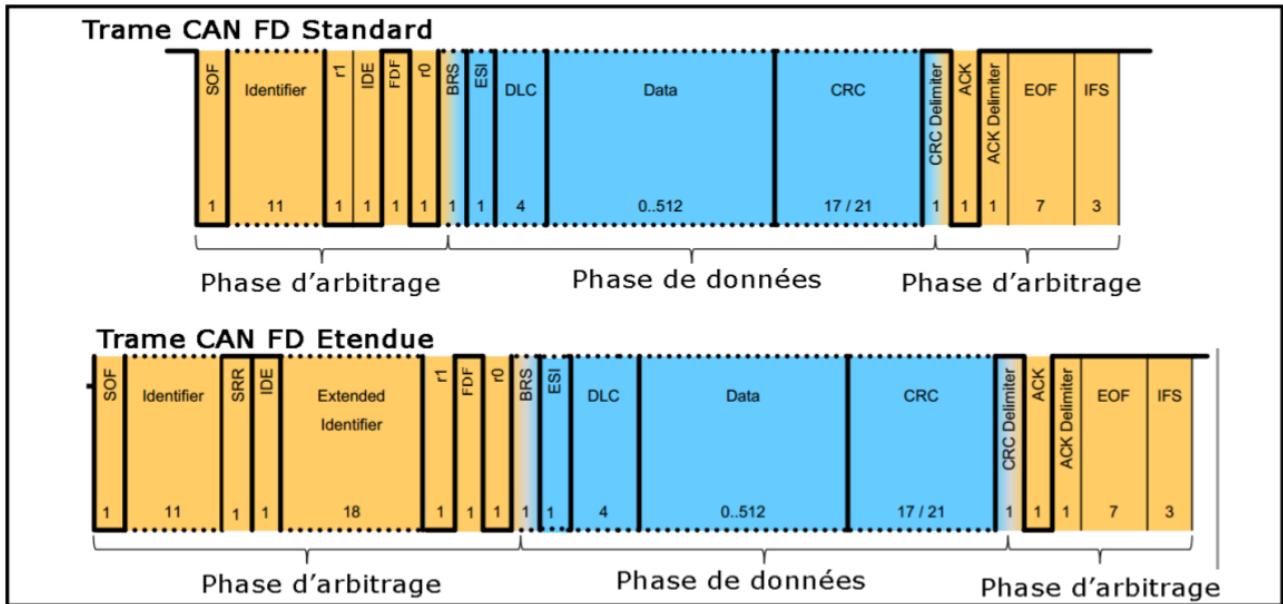


FIGURE 1.2 – Composition des trames ISO CAN FD

On a l'apparition des nouveaux champs par rapport à la trame CAN 2.0B :

- **FDF** : le CAN FD utilise le bit R0 (dominant) du CAN, le bit est nommé FDF et il est envoyé récessif.
- **R1 (dominant)** : remplace le bit RTR car il n'y a pas de trames de requête.
- **BRS** : contrôle le débit de la phase de donnée, dominant (pas de changement du débit), ou récessif (augmentation du débit dans la phase de donnée).
- **ESI** : indique l'état de la station, dominant (la station est en mode 'Error-Active'), ou récessif (la station est en mode 'Error-Passive').

## 3 Différence entre les trames CAN2.0B et ISO CANFD [3]

### 3.1 Taille de la trame

Dans le CAN, le champ de donnée a une taille maximale de 8 octets, ce qui fait que les trames peuvent avoir une taille maximale de 160 bits.

Dans le CAN FD, le champs de donnée peut avoir jusqu'à 64 octets de données. La taille de la trame peut donc atteindre 736 bits.

⇒ Le CAN FD permet donc d'éviter le fractionnement des longues trames.

## 3.2 Utilisation de la bande passante

Dans le CAN, seul 40-50% de la bande passante est utilisée pour transférer des données utiles :

- **Trame standard** : maximum 64 bits de données dans une trame de 135 bits  $\Rightarrow$  48% d'utilisation de la bande passante.
- **Trame étendue** : maximum 64 bits de données dans une trame de 160 bits  $\Rightarrow$  40% d'utilisation de la bande passante.

Dans le CAN FD, utilisation de jusqu'à 70% de la bande passante pour transférer des données utiles :

- **Trame standard** : maximum 512 bits de données dans une trame de 712 bits  $\Rightarrow$  72% d'utilisation de la bande passante.
- **Trame étendue** : maximum 512 bits de données dans une trame de 736 bits  $\Rightarrow$  70% d'utilisation de la bande passante.

$\Rightarrow$  Le CAN FD permet une meilleure utilisation de la bande passante.

## 3.3 Vitesse de transmission

Dans le CAN, la vitesse de transmission est limitée pour garantir la propriété de l'arbitrage non destructif, on peut donc atteindre une vitesse maximale de 1Mb/s.

Dans le CAN FD, on peut augmenter la vitesse de transmission dans le champ data, ce qui permet d'atteindre des vitesses plus grandes selon la charge de la trame :

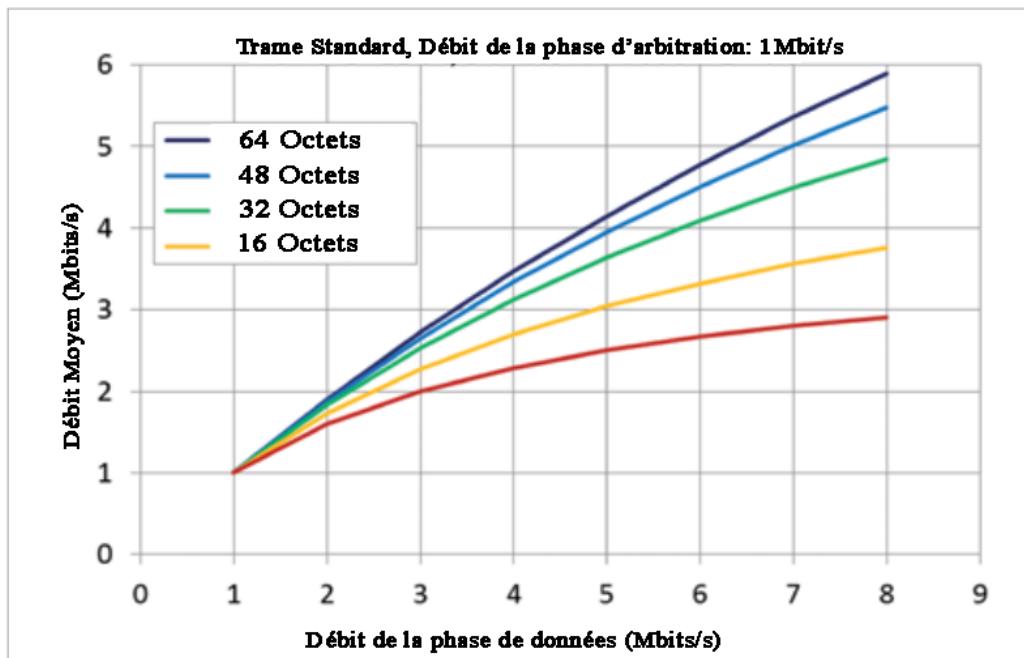


FIGURE 1.3 – Les vitesses de transmission du CAN FD selon la charge

### 3.4 Calcul du CRC

Le CRC est utilisé par le CAN et le CAN FD pour détecter les erreurs lors de la transmission de la trame. L'émetteur calcule le CRC de la trame (entre SOF et le dernier bit DATA) et l'envoie dans le champ CRC. Toutes les autres stations reçoivent la trame et calculent à leurs tournes le CRC pour le comparer avec le CRC reçu. Si une seule station déclare une erreur CRC, la trame est considérée corrompue par toutes les autres stations.

Pour le CAN on a :

- Le polynôme générateur utilisé est :  $g(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ , il a une distance de Hamming de 6, ce qui permet de détecter jusqu'à 5 *bit flips*.
- La valeur initiale du registre CRC est : '00...0'.
- Le *bit stuffing* se fait sur la trame complète (après calcul du CRC), ce qui peut poser des problèmes de *bit flip*.

Pour le CAN FD on a :

- Utilisation de deux polynômes générateurs de distance de Hamming de 6 :  $g(x) = x^{17} + x^{16} + x^{14} + x^{13} + x^{11} + x^6 + x^4 + x^3 + x^1 + 1$  et  $g(x) = x^{21} + x^{20} + x^{13} + x^{11} + x^7 + x^4 + x^3 + x^1 + 1$ .
- La valeur initiale du registre CRC est : '10...0'.
- Le *bit stuffing* se fait sur la trame avant le calcul du CRC.
- Ajout du champ *Stuff Count* pour éviter le problème de raccourcissement ou rallongement de trame.
- *Bit stuffing* fixe du champ CRC.

⇒ Le CAN FD introduit des mécanismes plus fiables pour le calcul du CRC, ce qui permet d'éviter le problème du *bit flip* du CAN.

# Chapitre 2

## Configuration du bit time

Le but de ce chapitre est de configurer le *bitTime* dans la phase d’arbitrage et la phase data. La configuration consiste d’abord en la définition de la longueur du *bitTime* et des différents segments [voir section 1.2, page 17], puis en la définition du point d’échantillonnage et enfin la définition de la longueur du SJW, qui est utilisé pour la synchronisation.

# 1 Composition du bit time

## 1.1 Définitions [2]

- Le *Minimum Time Quantum* est la plus petite unité de temps. Il est donné par l'horloge de l'oscillateur du contrôleur CAN FD selon l'équation :

$$\text{MinimumTimeQuantum} = t_{osc} = \frac{1}{f_{CANFD}} \quad (2.1)$$

- Le *Time Quantum* est construit à partir du *Minimum Time Quantum* en multipliant par le pré-diviseur comme suit :

$$\text{NominalTimeQuantum} = \text{MinimumTimeQuantum} \cdot \text{NominalPrescaler} \quad (2.2a)$$

$$\text{DataTimeQuantum} = \text{MinimumTimeQuantum} \cdot \text{DataPrescaler} \quad (2.2b)$$

- Le *Bit Time* est la longueur du bit (en s), il est calculé à partir du *bit rate* comme suit :

$$t_{NominalBitTime} = \frac{1}{\text{NominalBitRate}} \quad (2.3a)$$

$$t_{DataBitTime} = \frac{1}{\text{DataBitRate}} \quad (2.3b)$$

- Le *Bit Time Quantum* est calculé à partir du *bit time* en divisant sur le *TimeQuantum* :

$$\text{NominalBitTimeQuantum} = \frac{t_{NominalBitTime}}{\text{NominalTimeQuantum}} \quad (2.4a)$$

$$\text{DataBitTimeQuantum} = \frac{t_{DataBitTime}}{\text{DataTimeQuantum}} \quad (2.4b)$$

- La *Tolérance d'horloge* d'une station  $f_{osc}$  autour de la valeur nominal  $f_{nom}$  est définie par :

$$(1 - df) \cdot f_{nom} \leq f_{osc} \leq (1 + df) \cdot f_{nom} \quad (2.5)$$

df est la tolérance maximale de la station. Par la suite, on va travailler par la valeur de la tolérance en PPM :  $\text{Tolerance} = df \times 10^6$ .

## 1.2 La composition du bit time

La figure 2.1 montre la décomposition du bit time du CAN FD :

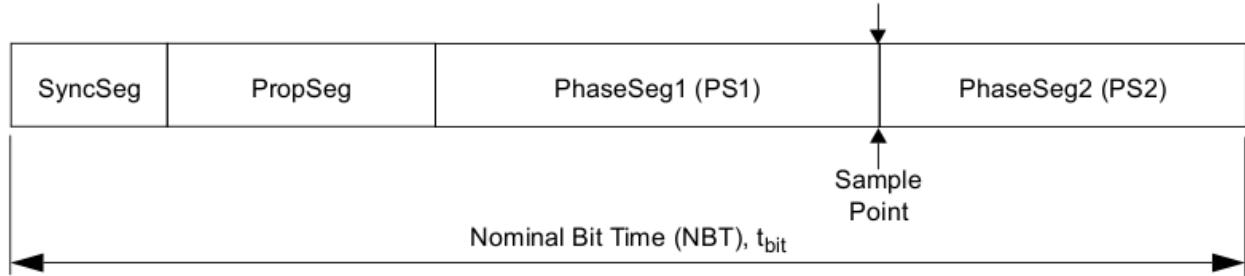


FIGURE 2.1 – CAN FD bit time [1]

- *Le segment de synchronisation (Synchronization Segment)* : utilisé pour synchroniser les récepteurs sur l'émetteur, sa durée est fixée à  $1 \cdot TQ$ .
- *Le segment de propagation (Propagation Segment)* : utilisé pour compenser le retard de transmission dans le bus.
- *Les segments de phase 1 et 2 (Phase Seg 1,2)* : utilisé pour la synchronisation, ils peuvent être raccourcis ou rallongés par la valeur SJW.
- *SJW* : est la valeur maximale par laquelle on peut rallonger (PS1) ou raccourcir (PS2).

## 1.3 Les valeurs des paramètres du bit time

la gamme des valeurs des paramètres du bit time est donnée dans le tableau suivant :

Paramètres	CAN	CAN FD	
	Nominal bit time	Nominal bit time	Data bit time
Pré-diviseur	1-32 NTQ	1-32 NTQ	1-32 DTQ
BitTimeQuantum	8-25 NTQ	8-80 NTQ	5-25 DTQ
Seg de synchronisation	1 NTQ	1 NTQ	1 DTQ
Seg de propagation	1-8 NTQ	1-48 NTQ	0-8 DTQ
Seg de la phase 1	1-8 NTQ	1-16 NTQ	1-8 DTQ
Seg de la phase 2	2-8 NTQ	2-16 NTQ	2-8 DTQ
SJW	1-4 NTQ	1-16 NTQ	1-8 DTQ

TABLE 2.1 – La taille des paramètres du bit time [2]

## 2 Configuration du bit time

### 2.1 Configuration du nominal bit time

#### Étape 1 : Calculer le pré-diviseur et le bit time quantum

La 1<sup>re</sup> étape dans la configuration du bit time est de choisir le pré-diviseur et le bit time quantum. La relation entre les deux est donnée par l'équation suivante :

$$\text{D'après (2.4a)} \quad \text{NominalBitTimeQuantum} = \frac{t_{bit}}{\text{NominalTimeQuantum}}$$

$$\text{Et d'après (2.2a)} \quad \text{NominalTimeQuantum} = \frac{\text{NominalPrescaler}}{f_{CANFD}}$$

$$\begin{aligned} \text{Donc } \text{NominalBitTimeQuantum} &= \frac{t_{bit}}{\text{NominalTimeQuantum}} \\ &= \frac{t_{bit} \cdot f_{CANFD}}{\text{NominalPrescaler}} \\ &= \frac{f_{CANFD}}{\text{NominalPrescaler} \cdot f_{NominalBitRate}} \end{aligned} \tag{2.6}$$

La méthode qu'on va utiliser pour déterminer le *NominalPrescaler* et le *NominalBitTimeQuantum* est de fixer à chaque fois le *NominalBitTimeQuantum* et calculer le *NominalPrescaler* par l'équation (2.6).

Pour chaque couple, on va calculer la valeur d'erreur qui est définie par : “*erreur* =  $f_{CANFD} - NominalBitRate \cdot NominalPrescaler \cdot NominalBitTimeQuantum$ ” et choisir la ou les configurations qui ont la plus petite erreur.

Dans le cas où plusieurs configurations ont la même meilleure erreur, on va choisir la configuration dont le *NominalBitTimeQuantum* est le plus grand pour plus de précision.

L'algorithme utilisé est le suivant :

---

#### Algorithm 2.1 Calcul du *NominalPrescaler* et *NominalBitTimeQuantum*

---

- 1:  $\text{NominalBTQ} \leftarrow 80$
  - 2:  $\text{tempNominalBTQ} \leftarrow 80$
  - 3:  $\text{error} \leftarrow \text{UINT32\_MAX}$
  - 4:  $\text{tempError} \leftarrow \text{UINT32\_MAX}$
  - 5:  $\text{NominalPrescaler} \leftarrow 32$
  - 6:  $\text{tempNominalPrescaler} \leftarrow \frac{f_{CANFD}}{\text{NominalBTQ} \cdot \text{NominalBitRate}}$
-

---

```

7: while  $tempNominalBTQ \geq 8$  and  $tempNominalPrescaler \leq 32$  do
8:   if  $tempNominalPrescaler > 0$  then
9:      $tempError \leftarrow (\frac{f_{CANFD}}{tempNominalBTQ \cdot tempNominalPrescaler}) - NominalBitRate$ 
10:    if  $tempError < error$  then
11:      erreur  $\leftarrow tempError$ 
12:       $NominalBTQ \leftarrow tempNominalBTQ$ 
13:       $NominalPrescaler \leftarrow tempNominalPrescaler$ 
14:    end if
15:     $tempNominalBTQ--$ 
16:     $tempNominalPrescaler = \frac{f_{CANFD}}{NominalBTQ \cdot NominalBitRate}$ 
17:  end if
18: end while

```

---

## Étape 2 : Calculer le segment de propagation minimal

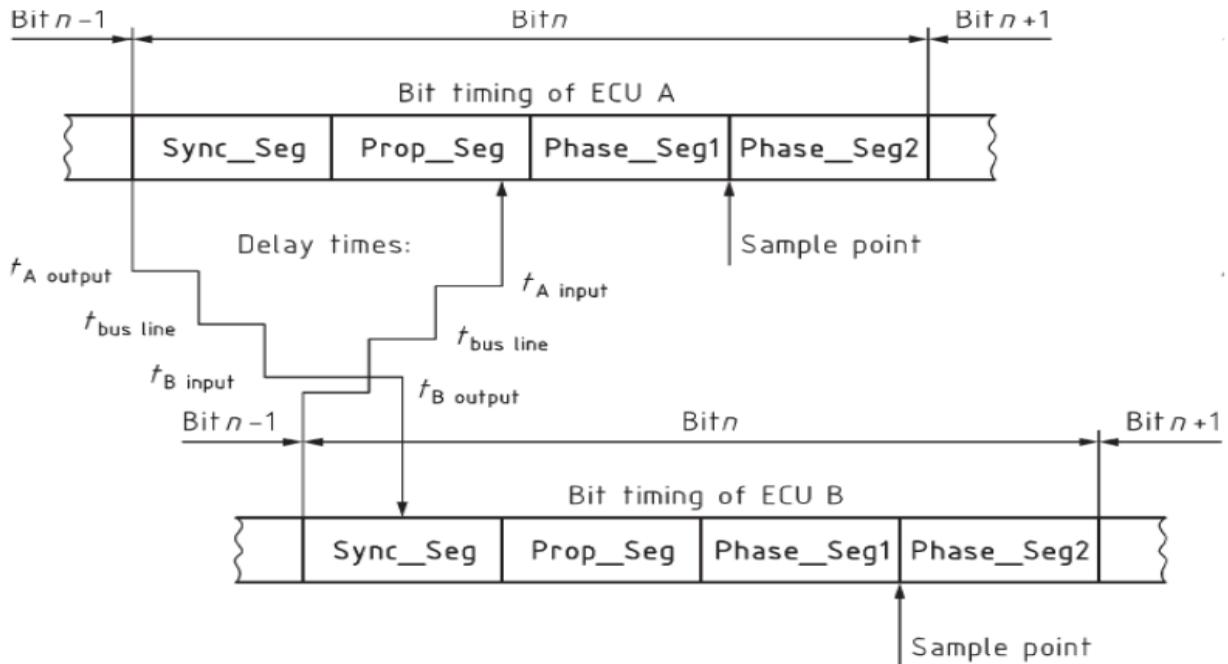


FIGURE 2.2 – Le temps de propagation entre deux stations CAN [2]

Le temps de propagation minimal doit être supérieur à deux fois le temps entre les deux stations les plus lointaines du bus, afin d'assurer l'arbitrage non destructif du CAN.

- **Le temps de retard** d'une station CAN  $t_{node}$  est la somme de tous les retards asynchrones qui se passent lors de la transmission et la réception d'une trame :

$$t_{node} = t_{output} + t_{input}$$

$t_{output}$  est le temps nécessaire pour que le bit sorte du contrôleur vers le bus CAN FD.  
 $t_{input}$  est le temps entre pour que le bit arrive du bus vers le contrôleur CAN FD.

- **Le temps de propagation** entre les deux extrémités du réseau  $t_{busLine}$  est calculé par l'équation :

$$t_{busLine} = \delta \cdot L$$

$\delta$  (specific line delay) est la retard de propagation dans la ligne.

$L$  est la longueur maximale du réseau.

- **Le temps du segment de propagation minimal** est la somme du *temps de propagation* entre les deux extrémités du bus et le *temps de retard* des deux stations :

$$t_{propagationSegmentMin} = t_{nodeA} + t_{nodeB} + 2 \cdot t_{busLine} \quad (2.7)$$

- **Le NominalPropagationSegment** se calcule par l'équation suivante :

$$\begin{aligned} NominalPropagationSegment &= \left\lceil \frac{t_{propagationSegment}}{NominalTimeQuantum} \right\rceil \\ &= \left\lceil t_{propagationSegment} \cdot \frac{f_{CANFD}}{NominalPrescaler} \right\rceil \end{aligned} \quad (2.8)$$

### Étape 3 : Calcul du $NominalSJW_{min}$

D'après l'équation (3) de la tolérance de l'horloge du [2], on a la condition :

$$df < \frac{NominalSJW}{2 \cdot 10 \cdot NominalBTQ}, \text{ ce qui donne :}$$

$$NominalSJW \geq \lceil 20 \cdot df \cdot NominalBTQ \rceil = NominalSJW_{min} \quad (2.9)$$

Le calcul du  $NominalSJW_{min}$  va nous permettre de définir les limites sur  $NominalPhaseSeg1$  et  $NominalPhaseSeg2$ .

### Étape 4 : Configuration du NominalBitTime

On va commencer par tester si le  $NominalPropagationSegment$  est trop long :

(( $NominalPropSeg > NominalBTQ - (1 + 3)$ )), dans ce cas, on va renvoyer un code d'erreur et donner la bonne configuration la plus proche quand même, et c'est à l'utilisateur de choisir s'il veut procéder avec la configuration ou non.

Si le  $NominalPropagationSegment$  est de taille normale, on va tester si le  $NominalSJW$  est supérieur au  $NominalSJW_{min}$  ou non. Pour faire le test, on va supposer que les deux segments de phase 1 et 2 ont une longueur égale à  $NominalSJW_{min}$  (la taille minimal).

Dans le cas où  $NominalSJW < NominalSJW_{min}$  alors on ne va pas assurer à 100% que la **resynchronisation** va permettre de synchroniser correctement le récepteur sur l'émetteur. Comme dans le cas précédent, on va envoyer un code d'erreur et c'est à l'utilisateur de décider s'il veut procéder avec la configuration ou non.

Finalement, si  $NominalSJW \geq NominalSJW_{min}$  alors on va configurer le  $NominalBitTime$  et renvoyer un code de succès. On va calculer l'intervalle possible pour le  $NominalPhaseSeg2$  en tenant compte des conditions suivantes :

### NominalPhaseSeg2<sub>min</sub> :

- La longueur minimal du PhaseSeg2 est 2 donc :  $NominalPhaseSeg2 \geq 2$
- On a :  $NominalPhaseSeg2 \geq NominalSJW_{min}$
- On a la condition (voir Annexe A, équation A.1) :

$$NominalPhaseSeg2 \geq \left\lceil \frac{26 \cdot NominalBTQ \cdot df}{1 + 2 \cdot df} \right\rceil$$

### NominalPhaseSeg2<sub>max</sub> :

- La longueur maximal du PhaseSeg2 correspond à la valeur minimal du PhaseSeg1, or  $NominalPhaseSeg1 \geq 1$  donc :

$$NominalPhaseSeg2 \leq [NominalBTQ - (1 + NominalPropSeg)] - 1$$

- On a  $NominalPhaseSeg1 \geq NominalSJW_{min}$  donc :

$$NominalPhaseSeg2 \leq [NominalBTQ - (1 + NominalPropSeg)] - NominalSJW_{min}$$

- On a la condition (voir Annexe A, équation A.2) :

$$NominalPhaseSeg2 \leq \left\lfloor \frac{(NominalBTQ - (1 + NominalPropSeg)) - 26 \cdot NominalBTQ \cdot df}{1 - 2 \cdot df} \right\rfloor$$

Le calcul des deux segments de phase 1 et 2 va se faire d'une manière à avoir le  $NominalSJW$  le plus grand possible. L'algorithme utilisé est le suivant :

---

#### Algorithm 2.2 Configuration du *NominalBitTime*

---

```

1: ErrorCode ← 0
2: if  $(NominalBTQ - (1 + NominalPropSeg)) < 3$  or  $NominalPropSeg > 48$  then
3:    $NominalPhaseSeg1 \leftarrow \min(16, \max(1, NominalSJW_{min}))$ 
4:    $NominalPhaseSeg2 \leftarrow \min(16, \max(2, NominalSJW_{min}))$ 
5:    $NominalPropSeg \leftarrow NominalBTQ - (1 + NominalPhaseSeg1 + NominalPhaseSeg2)$ 
6:   ErrorCode ← ErrorCode + 0x10

7: else
8:   halfPhaseSeg ←  $\frac{NominalBTQ - (1 + NominalPropSeg)}{2}$ 
9:   if  $(NominalBTQ - (1 + NominalPropSeg)) < 2 \cdot NominalSJW_{min}$  then
10:     $NominalPhaseSeg2 \leftarrow \min(16, \max(2, halfPhaseSeg))$ 
11:     $NominalPhaseSeg1 \leftarrow \min(16, NominalBTQ - (1 + NominalPropSeg + NominalPhaseSeg2))$ 
12:     $NominalPropSeg \leftarrow NominalBTQ - (1 + NominalPhaseSeg1 + NominalPhaseSeg2)$ 
13:    ErrorCode ← ErrorCode + 0x04

```

---

---

```

14:   else
15:      $NPS2min \leftarrow \left\lceil \frac{26 \cdot NominalBTQ \cdot df}{1+2 \cdot df} \right\rceil$ 
16:     if  $NPS2min > halfPhaseSeg$  then
17:        $NPS2min \leftarrow 0$ 
18:     end if
19:      $NominalPS2_{min} \leftarrow \max(2, NominalSJW_{min}, NPS2min)$ 

20:     $NPS2max1 \leftarrow (NominalBTQ - (1 + NominalPropSeg)) - 1$ 
21:     $NPS2max2 \leftarrow (NominalBTQ - (1 + NominalPropSeg)) - NominalSJW_{min}$ 
22:     $NPS2max3 \leftarrow \left\lfloor \frac{(NominalBTQ - (1 + NominalPropSeg)) - 26 \cdot NominalBTQ \cdot df}{1-2 \cdot df} \right\rfloor$ 
23:    if  $NPS2max3 < halfPhaseSeg$  or  $NPS2max3 < 0$  then
24:       $NPS2max3 \leftarrow 16$ 
25:    end if
26:     $NominalPS2_{max} \leftarrow \min(16, NPS2max1, NPS2max2, NPS2max3)$ 

27:    if  $NominalPS2_{max} \leq halfPhaseSeg$  then
28:       $NominalPhaseSeg2 \leftarrow NominalPS2_{max}$ 
29:    else if  $NominalPS2_{min} \geq halfPhaseSeg$  then
30:       $NominalPhaseSeg2 \leftarrow NominalPS2_{min}$ 
31:    else
32:       $NominalPhaseSeg2 \leftarrow halfPhaseSeg$ 
33:    end if

34:     $NominalPhaseSeg1 \leftarrow \min(16, NominalBTQ - (1 + NominalPropSeg) + NominalPhaseSeg2))$ 
35:     $NominalPropSeg \leftarrow \frac{NominalBTQ - (1 + NominalPhaseSeg1) + NominalPhaseSeg2)}{2}$ 
36:     $NominalSamplePoint_{min} \leftarrow \frac{NominalBTQ - NominalPS2_{max}}{NominalBTQ} \cdot 100$ 
37:     $NominalSamplePoint_{max} \leftarrow \frac{NominalBTQ - NominalPS2_{min}}{NominalBTQ} \cdot 100$ 
38:  end if
39: end if
40:  $NominalSamplePoint \leftarrow \frac{NominalBTQ - NominalPhaseSeg2}{NominalBTQ} \cdot 100$ 

```

---

### Remarques :

- 01 : Le *ErrorCode* est un mot de 8 octets, chaque octet signale un type d'erreur [voir Annexe E, page 60].
- 02 : La taille minimal des deux segments de phase est 3 (1 pour *NominalPhaseSeg1* et 2 pour *NominalPhaseSeg2*).
- 02-06 : Cas où le segment de propagation est trop grand
- 09-13 : Cas où :  $NominalSJW < NominalSJW_{min}$
- 14-37 : Cas normal.
- 27-33 :  $NominalPhaseSeg2 \in [NominalPS2_{min}, NominalPS2_{max}]$ , on va chercher la valeur la plus proche au *halfPhaseSeg* pour avoir le *NominalSJW* maximal.

## Étape 5 : Vérifier la tolérance de l'horloge

Après avoir configuré le *NominalBitTime*, on va vérifier si la fréquence fourni vérifie l'équation 2.5 :

---

### Algorithm 2.3 Vérification de la tolérance d'horloge du NominalBitTime

---

```

1: NominalDiff  $\leftarrow \frac{|f_{CANFD} - NominalBitRate \cdot NominalBTQ \cdot NominalBP|}{f_{CANFD}} \cdot 10^6$ 
2: if NominalDiff > Tolerance then
3:   ErrorCode  $\leftarrow ErrorCode + 0x01$ 
4: end if
```

---

**Remarque :** On multiplie par 1.000.000 parce que la tolérance max de l'horloge est exprimée en PPM.

## 2.2 Configuration du data bit time

### Étape 1 : Calculer le pré-diviseur et le bit time quantum

Comme pour le *NominalBitTime*, on va commencer par choisir le pré-diviseur et le bit time quantum. La relation entre les deux est donnée par l'équation suivante :

$$DataBitTimeQuantum = \frac{f_{CANFD}}{DataPrescaler \cdot f_{DataBitRate}} \quad (2.10)$$

L'algorithme utilisé est le suivant :

---

### Algorithm 2.4 Calcul du *DataPrescaler* et *DataBitTimeQuantum*

---

```

1: DataBTQ  $\leftarrow 25$ 
2: tempDataBTQ  $\leftarrow 25$ 
3: error  $\leftarrow UINT32\_MAX$ 
4: tempError  $\leftarrow UINT32\_MAX$ 
5: DataPrescaler  $\leftarrow 32$ 
6: tempDataPrescaler  $\leftarrow \frac{f_{CANFD}}{DataBTQ \cdot DataBitRate}$ 

7: while tempDataBTQ  $\geq 5$  and tempDataPrescaler  $\leq 32$  do
8:   if tempDataPrescaler > 0 then
9:     tempError  $\leftarrow \left( \frac{f_{CANFD}}{tempDataBTQ \cdot tempDataPrescaler} \right) - DataBitRate$ 
10:    if tempError < error then
11:      erreur  $\leftarrow tempError$ 
12:      DataBTQ  $\leftarrow tempDataBTQ$ 
13:      DataPrescaler  $\leftarrow tempDataPrescaler$ 
14:    end if
15:    tempDataBTQ--
16:    tempDataPrescaler  $\leftarrow \frac{f_{CANFD}}{DataBTQ \cdot DataBitRate}$ 
17:  end if
18: end while
```

---

## Étape 2 : Calcul du $DataSJW_{min}$

D'après l'équation (5) de la tolérance d'horloge du [2], on a la condition :  
 $df < \frac{DataSJW}{2 \cdot 10 \cdot DataBTQ}$ , ce qui donne :

$$DataSJW \geq \lceil 20 \cdot df \cdot DataBTQ \rceil = DataSJW_{min} \quad (2.11)$$

Le calcul du  $DataSJW_{min}$  va nous permettre de définir les limites sur  $NominalPhaseSeg1$  et  $NominalPhaseSeg2$ .

## Étape 3 : Configuration du DataBitTime

De la même manière que pour la configuration du  $NominalBitTime$ , on va configurer le  $DataBitTime$ , la seule différence c'est que la taille du  $DataPropSeg$  est initialement à 0. On aura pas donc le cas où le segment de propagation est trop grand.

On va calculer l'intervalle possible pour le  $DataPhaseSeg2$  en tenant compte des conditions suivantes :

### $DataPhaseSeg2_{min}$ :

- La longueur minimal du PhaseSeg2 est 2 donc :  $DataPhaseSeg2 \geq 2$
- On a :  $DataPhaseSeg2 \geq DataSJW_{min}$
- On a la condition (voir Annexe A, équation A.3) :

$$DataPhaseSeg2 \geq \left\lceil \frac{2.df.(2.NominalBTQ - NominalPhaseSeg2). \frac{NominalPrescaler}{DataPrescaler} + 8.df.DataBTQ - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1)}{(1 - 2.df)} \right\rceil$$

### $DataPhaseSeg2_{max}$ :

- La longueur maximal du PhaseSeg2 correspond à la valeur minimal du PhaseSeg1, or  $DataPhaseSeg1 \geq 1$  donc :

$$DataPhaseSeg2 \leq (DataBTQ - 1) - 1$$

- On a  $DataPhaseSeg1 \geq DataSJW_{min}$  donc :

$$DataPhaseSeg2 \leq (DataBTQ - 1) - DataSJW_{min}$$

- On a la condition (voir Annexe A, équation A.4) :

$$DataPhaseSeg2 \leq \left\lceil \frac{DataBTQ - \left[ 1 + 8.df.DataBTQ + \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) + 2.df.(2.NominalBTQ - NominalPhaseSeg2). \frac{NominalPrescaler}{DataPrescaler} \right]}{1 + 2.df} \right\rceil$$

Le calcul des deux segments de phase 1 et 2 va se faire d'une manière à avoir le *DataSJW* le plus grand possible. L'algorithme utilisé est le suivant :

---

**Algorithm 2.5** Configuration du *DataBitTime*


---

```

1: if  $(DataBTQ - 1) < 2 \cdot DataSJW_{min}$  then
2:    $DataPhaseSeg2 \leftarrow \min(8, \max(2, \frac{DataBTQ-1}{2}))$ 
3:    $DataPhaseSeg1 \leftarrow \min(8, \max(1, DataBTQ - (1 + DataPhaseSeg2)))$ 
4:    $ErrorCode \leftarrow ErrorCode + 0x08$ 

5: else
6:    $halfPhaseSeg \leftarrow \frac{DataBTQ-1}{2}$ 
7:    $tempDPS2min \leftarrow \left\lfloor \frac{2.df.(2.NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} + 8.df.DataBTQ - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1)}{(1 - 2 \cdot df)} \right\rfloor$ 
8:   if  $tempDPS2min > halfPhaseSeg$  or  $tempDPS2min < 0$  then
9:      $tempDPS2min \leftarrow 0$ 
10:    end if

11:   $DataPS2_{min} \leftarrow \max(2, DataSJW_{min}, tempDPS2min)$ 

12:   $tempDPS2max1 \leftarrow (DataBTQ - 1) - 1$ 
13:   $tempDPS2max2 \leftarrow (DataBTQ - 1) - DataSJW_{min}$ 
14:   $tempDPS2max3 \leftarrow \left\lfloor \frac{DataBTQ - \left[ 1 + 8.df.DataBTQ + \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) + 2.df.(2.NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} \right]}{1 + 2 \cdot df} \right\rfloor$ 
15:  if  $tempDPS2max3 < halfPhaseSeg$  or  $tempDPS2min < 0$  then
16:     $tempDPS2max3 \leftarrow 8$ 
17:  end if

18:   $DataPS2_{max} \leftarrow \min(8, tempDPS2max1, tempDPS2max2, tempDPS2max3)$ 

```

---

---

```

19: if  $DataPS2_{max} \leq halfPhaseSeg$  then
20:    $DataPhaseSeg2 \leftarrow DataPS2_{max}$ 
21: else if  $DataPS2_{min} \geq halfPhaseSeg$  then
22:    $DataPhaseSeg2 \leftarrow DataPS2_{min}$ 
23: else
24:    $DataPhaseSeg2 \leftarrow halfPhaseSeg$ 
25: end if

26:  $DataPhaseSeg1 \leftarrow min(8, max(1, DataBTQ - (1 + DataPropSeg + DataPhaseSeg2)))$ 

27: end if

28:  $DataPropSeg \leftarrow DataBTQ - (1 + DataPhaseSeg1 + DataPhaseSeg2)$ 

29:  $DataSamplePoint_{min} \leftarrow \frac{DataBTQ - DataPS2_{max}}{DataBTQ} \cdot 100$ 
30:  $DataSamplePoint_{max} \leftarrow \frac{DataBTQ - DataPS2_{min}}{DataBTQ} \cdot 100$ 
31:  $DataSamplePoint \leftarrow \frac{DataBTQ - DataPhaseSeg2}{DataBTQ} \cdot 100$ 

```

---

#### Étape 4 : Vérifier la tolérance de l'horloge

Après avoir configuré le *DataBitTime*, on va vérifier si la fréquence fournie vérifie l'équation 2.5 :

---

##### **Algorithm 2.6** Vérification de la tolérance d'horloge du DataBitTime

---

```

1:  $DataDiff \leftarrow \frac{|f_{CANFD} - DataBitRate \cdot DataBTQ \cdot DataBP|}{f_{CANFD}} \cdot 10^6$ 
2: if  $DataDiff > Tolerance$  then
3:    $ErrorCode \leftarrow ErrorCode + 0x02$ 
4: end if

```

---

**Remarque :** On multiplie par 1.000.000 parce que la tolérance max de l'horloge est exprimée en PPM.

# Chapitre 3

## La bibliothèque MCP2517FD

Dans ce chapitre, nous allons voir la composition de la bibliothèque MCP2517FD. Nous allons commencer par les méthodes de configuration du contrôleur MCP2517FD. Par la suite, nous allons voir les méthodes d'envoi et de réception des trames, et finalement, nous allons donner un exemple d'une cas d'utilisation de la bibliothèque en mode *LoopBack*.

# 1 Introduction au contrôleur [4]

Le **MCP2517FD** est un contrôleur CAN FD avec une interface SPI fabriqué par la société **Microship**. Le contrôleur est conforme à la norme *ISO11898-1 :2015*, et supporte les trames CAN 2.0B et CAN FD.

Les caractéristiques du contrôleur sont :

- La vitesse de la phase data peut atteindre jusqu'à 8Mb/s.
- La vitesse de l'interface SPI peut atteindre 20MHz.
- L'ordre de transmission est selon la priorité du canal de transmission.
- 31 FIFOs configurables pour la transmission ou la réception (transmission dans le FIFO par ordre d'arrivé).
- 1 canal de transmission TxQ (priorité de transmission aux messages avec le plus petit ID).
- Horodatage des messages transmis (enregistrer les informations du message dans la TEF).
- 32 filtres configurables.

# 2 Introduction à la bibliothèque

La bibliothèque MCP2517FD permet à l'utilisateur soit de configurer le contrôleur (configurer le bitTime, le mode de configuration, les canaux de transmissions, ...), soit d'envoyer des trames dans le bus ou de les recevoir.

Le code source complet de la bibliothèque est valable sous *Github* (lien dans la bibliographie [5]).

Les diagrammes UML suivants sont faits par le logiciel *Modelio*[6], et décrivent la composition de la bibliothèque :

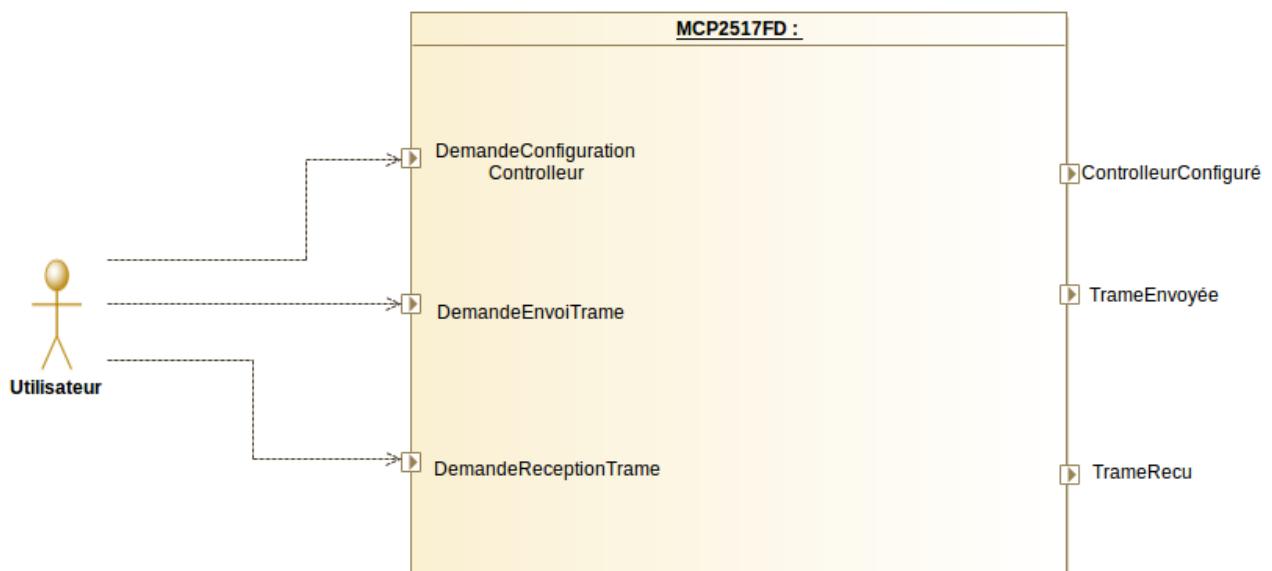


FIGURE 3.1 – Le diagramme de contexte du MCP2517FD

Les différentes fonctionnalités de la bibliothèque sont données par le diagramme de blocs suivant :

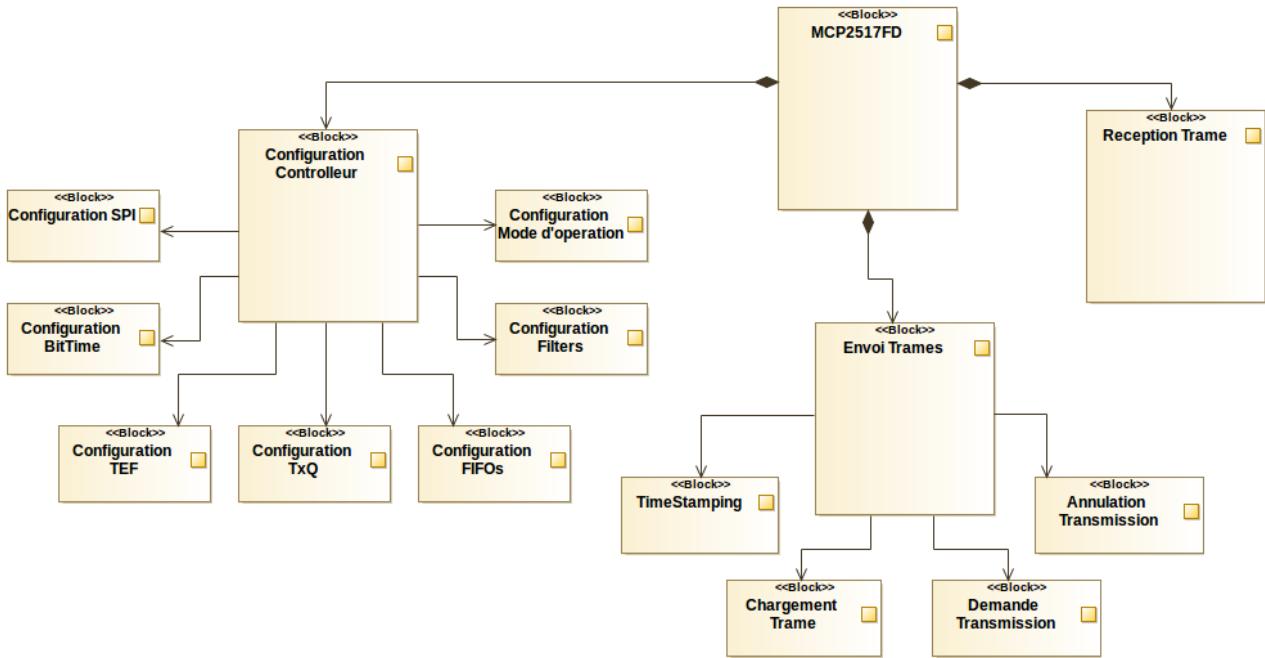


FIGURE 3.2 – Le diagramme de blocs du MCP2517FD

### 3 Diagramme de classe

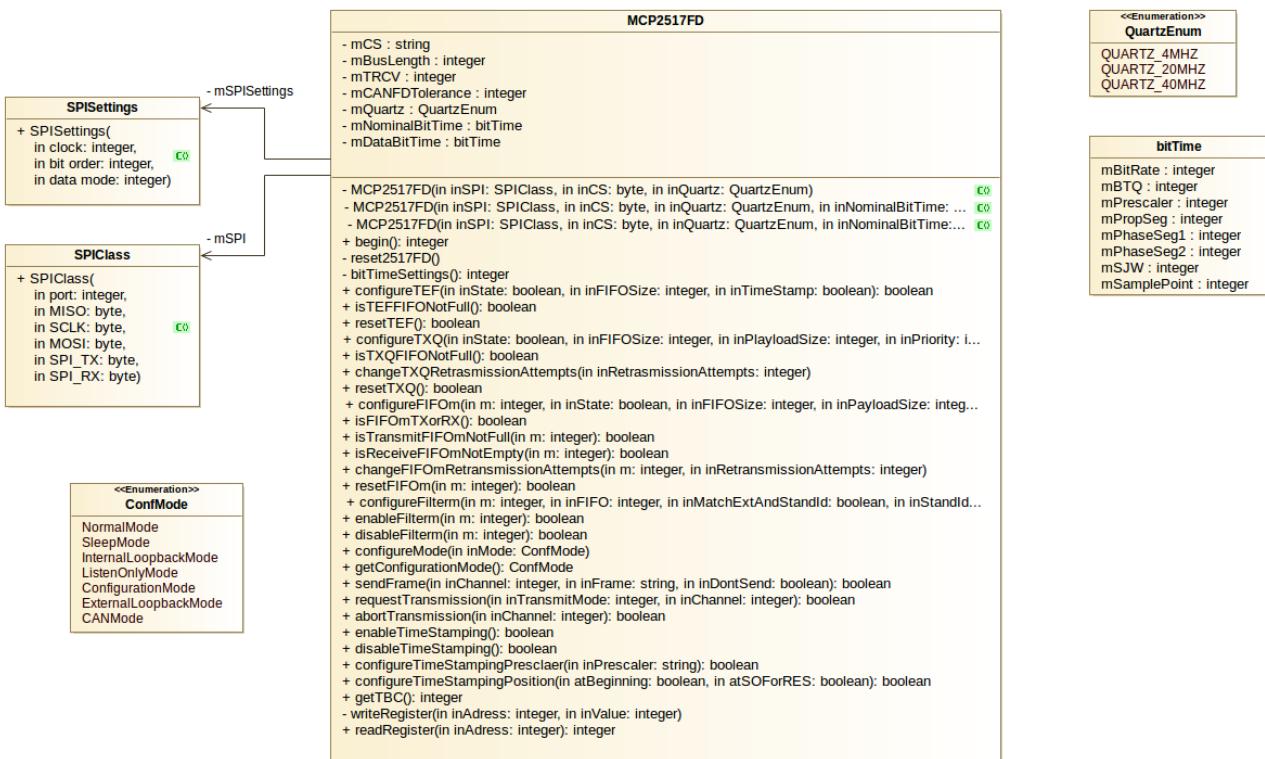


FIGURE 3.3 – Le diagramme de classe du MCP2517FD

### 3.1 Les attributs de la classe

- **mCs** : le port *Slave Select* de la communication SPI.
- **mSPI** : objet de la classe *SPIClass*, contient les autres ports SPI (MOSI, MISO, SCLK).
- **mSPISettings** : objet de la classe *SPISettings*, utilisé pour définir la vitesse de transmission SPI et pour envoyer les données.
- **mBusLength** : la longueur du bus (en m).
- **mTRCV** : le temps de retard dans le bus (en ns).
- **mCANFDTolerance** : la tolérance de l'horloge du contrôleur (en PPM).
- **mQuartz** : la vitesse de l'horloge du contrôleur.
- **mNominalBitTime** : contient les différents informations sur le nominal bitTime.
- **mDataBitTime** : contient les différents informations sur le data bitTime.

### 3.2 Les constructeurs de la classe

Il existe trois constructeurs pour la classe MCP2517FD, le constructeur principale est le suivant :

Pour les deux autres constructeurs, les valeurs non entrées correspondent aux valeurs par défaut (inNominalBitTime = 500Kb/s, inDataBitTime = 1Mb/s, inNominalSamplePoint = 0, inDataSamplePoint = 0, inBusLength = 40m, inTRCV = 80ns, inCANFDTolerance = 30PPM).

+ **MCP2517FD(**  
in inSPI: SPIClass,  
in inCS: byte,  
in inQuartz: QuartzEnum,  
in inNominalBitTime: integer,  
in inDataBitTime: integer,  
in inNominalSamplePoint: int...  
in inDataSamplePoint: integer,  
in inBusLength: integer,  
in inTRCV: integer,  
in inCANFDTolerance: integer)  


FIGURE 3.4 – Le constructeur par défaut de la classe MCP2517FD

## 4 Les méthodes de configuration

### 4.1 Configuration de la communication SPI

On commence par la déclaration d'un objet *SPIClass* qui prend comme paramètres les ports de la connexion SPI [7] :

```
1 SPIClass mySPI (&PERIPH_SPI, PIN_SPI_MISO, PIN_SPI_SCK, PIN_SPI_MOSI,  
PAD_SPI_TX, PAD_SPI_RX);
```

- **&PERIPH\_SPI** : l'adresse du port série utilisé pour la liaison SPI.
- **PIN\_SPI\_MISO** : port GPIO correspondant connecté au port MISO du contrôleur.
- **PIN\_SPI\_SCK** : port GPIO correspondant connecté au port SCK du contrôleur.
- **PIN\_SPI\_MOSI** : port GPIO correspondant connecté au port MOSI du contrôleur.
- **PAD\_SPI\_TX** : définit le multiplexage utilisé.

- **PAD\_SPI\_RX** : définit le multiplexage utilisé.

Par la suite on crée un objet de la classe MCP2517FD en utilisant l'un des constructeurs précédents :

```
1 MCP2517FD canController (mySPI, PIN_SPI_CS, inQuartz, inNominalBitTime,
    inDataBitTime, inNominalSamplePoint, inDataSamplePoint, inBusLength, inTRCV,
    inCANFDTolerance);
```

- **PIN\_SPI\_CS** : port GPIO correspondant connecté au port CS du contrôleur.

## 4.2 Configuration du bitTime

la méthode “*begin()*” permet de configurer la vitesse SPI, puis de configurer les bitTime selon les équations vu dans la section 2 :

```
1 uint32_t begin (void);
```

La fonction ne prend pas de paramètres et renvoie des codes d'erreurs (voir Annexe.E, page.60)

## 4.3 Configuration du mode d'opération

```
1 void configureMode (const ConfMode inMode);
```

Il existe 7 modes de configuration, déclarés en tant qu'énumération :

- **NormalMode** : le mode normale qui supporte les trames CAN 2.0B et ISO CANFD.
- **SleepMode** : le mode veille est un mode à faible consommation, où les valeurs des registres de la RAM sont conservées et l'horloge est éteinte.
- **InternalLoopbackMode** : le signal transmit est lié directement au pin Rx, et Tx est récessif.
- **ListenOnlyMode** : le contrôleur ne peut pas envoyer des trames.
- **ConfigurationMode** : utilisé lors de la configuration du contrôleur.
- **ExternalLoopbackMode** : le même que *InternalLoopbackMode*, mais les messages envoyées peuvent être lus dans le pin Tx.
- **CANMode** : supporte juste les trames CAN 2.0B.

<<Enumeration>>
ConfMode
NormalMode
SleepMode
InternalLoopbackMode
ListenOnlyMode
ConfigurationMode
External LoopbackMode
CANMode

FIGURE 3.5 – Les modes de configuration du contrôleur

**Remarque :** Après la configuration du bitTime, il faut entrer dans le *ConfigurationMode* pour configurer les filtres et les canaux de transmission.

## 4.4 Configuration du TXQ (Transmit Queue)

TxQ est utilisé pour envoyer des messages, l'envoi se fait selon l'ordre de l'ID des messages. Pour configurer TxQ, il faut utiliser la méthode :

```
1 bool configureTXQ (const bool inState = 1, const uint32_t inFIFOSize = 10, const  
    uint32_t inPayloadSize = 16, const uint32_t inPriority = 100);
```

- **inState** : activer ou désactiver TxQ.
- **inFIFOSize** : la taille du *buffer* du TxQ.
- **inPayloadSize** : la taille maximale du champ data.
- **inPriority** : la priorité du TxQ par rapport aux autres canaux de transmission ( $\in [0, 100]$ , 100 étant le plus prioritaire).

Pour changer le nombre de tentatives de retransmission (par défaut infini), il faut utiliser la méthode :

```
1 void changeTXQRetransmissionAttempts (const uint32_t inRetransmissionAttempts);
```

- **inRetransmissionAttempts = 0** : désactiver la retransmission.
- **inRetransmissionAttempts = 1** : 3 tentatives de retransmission.
- **inRetransmissionAttempts > 1** : retransmission infini.

## 4.5 Configuration des FIFOs

Il existe en total 31 FIFOs qui peuvent être utilisés en transmission ou réception. Chaque FIFO doit être configurée séparément en utilisant la méthode suivante :

```
1 bool configureFIFOm (const uint16_t m = 1, const bool inState = 1, const  
    uint32_t inFIFOSize = 10, const uint32_t inPayloadSize = 16, const uint32_t  
    inPriority = 0, const bool inTimeStamp = 0);
```

- **m** : la FIFO correspondante ( $\in [1, 31]$ ).
- **inState** : inState=1 => FIFO en mode transmission, inState=0 => FIFO en mode réception.
- **inFIFOSize** : la taille du *buffer* du FIFOm ( $\in [1, 32]$ ).
- **inPayloadSize** : la taille maximale du champs de données.
- **inPriority** : la priorité du FIFOm par rapport à TxQ et aux autres FIFOs de transmission.
- **inTimeStamp** : inTimeStamp=1 pour activer l'horodatage (voir section 4.8, page 34).

Pour changer le nombre de tentatives de retransmission (par défaut infini), il faut utiliser la méthode :

```
1 void changeFIFOmRetransmissionAttempts (const uint16_t m, const uint32_t  
    inRetransmissionAttempts);
```

## 4.6 Configuration TEF (Transmit Event FIFO)

TEF permet à l'application de garder l'ordre et le temps de transmission des messages. TEF stocke les messages envoyées (Juste les informations sur la trame et non le champ data). Pour configurer TEF, il faut utiliser la méthode :

```
1 bool configureTEF (const bool inState = 0, const uint32_t inFIFOSize = 1, const  
                      bool inTimeStamp = 0);
```

- **inState** : inState=1 pour utiliser TEF (réserver l'espace mémoire dans la RAM), inState=0 sinon.
- **inFIFOSize** : la taille du *buffer* du TEF ( $\in [1, 32]$ ).
- **inTimeStamp** : inTimeStamp=1 pour activer l'horodatage (voir section 4.8, page 34).

## 4.7 Configuration des filtres

Les trames reçus par le contrôleur ne sont pas enregistrées par défaut. Le rôle des filtres et de définir des règles pour enregistrer les trames. Il existe 32 filtres configurables dans le MCP2517FD.

Pour configurer un filtre, il faut utiliser la méthode suivante :

```
1 bool configureFilterm (const uint16_t m = 1, const uint32_t inFIFO = 1, const  
                        bool inMatchExtAndStandId = 1, const uint32_t inStandId = 0x000, const  
                        uint32_t inStandIdRange = 0x000, const uint32_t inExtId = 0x00000, const  
                        uint32_t inExtIdRange = 0x00000, const bool inMatchExtOrStandId = 0);
```

- **inFIFO** : la FIFO où on va enregistrer les trames qui valide le filtre (doit être FIFO de réception).
- **inMatchExtAndStandId** : si =1, alors il faut voir à la fois l'id standard et l'id étendue du message reçu, sinon si =0, alors juste l'un des deux (lequel voir inMatchExtOrStandId).
- **inMatchExtOrStandId** : si inMatchExtAndStandId=0 alors si inMatchExtOrStandId=0, il faut comparer juste l'id standard du message reçu sinon si inMatchExtOrStandId=1, il faut comparer l'id étendu.
- **inStandId, inStandIdRange** : l'id standard du message reçu doit être dans la gamme tel que si inStandIdRange[i]=1 alors StandId[i] du message reçu doit être le même que inStandId[i], sinon si inStandIdRange[i]=0 alors Stand[i] du message reçu peut prendre n'importe quelle valeur (voir exemple).
- **inExtId, inExtIdRange** : même chose que pour l'id standard.

Exemples :

```
1/ configureFilterm(1, 2, 1, 0x000, 0x000, 0x00000, 0x00000, 0)
```

- enregistrer les messages qui passent le filtre1 à FIFO2 (FIFO2 doit être de réception).  
→ voir les deux id à la fois (standard et étendu).

→ enregistrer tous les messages qui ont l'id standard entre 0x000 et 0xFFFF et l'id étendu entre 0x00000 et 0xFFFF (tous les messages).

## 2/ `configureFilterm(2, 5, 0, 0x000, 0xF00, 0x00000, 0x00000, 0)`

→ enregistrer les messages qui passent le filtre2 à FIFO5 (FIFO5 doit être de réception).

→ voir l'id standard seulement.

→ enregistrer tous les messages qui ont l'id standard entre 0x000 et 0x0FF.

## 3/ `configureFilterm(3, 10, 1, 0xA51, 0xFFF, 0x00B08, 0xFFFF, 0)`

→ enregistrer les messages qui passent le filtre3 à FIFO10 (FIFO10 doit être de réception).

→ voir les deux id à la fois (standard et étendu).

→ enregistrer tous les messages qui ont l'id standard : 0xA51 et l'id étendu : 0x00B08, c-a-d les messages avec l'id suivant : 0x00B08A51.

## 4.8 Configuration de l'horodatage (Time Stamping)

Pour horodater les trames envoyées, il faut d'abord activer l'horodatage dans les canaux de transmission souhaités.

Après, il faut configurer l'horodatage en suivant les étapes suivantes :

1/Désactiver l'horodatage en utilisant la méthode :

```
1 bool disableTimeStamping ( void );
```

2/Configurer le *time base counter prescaler* en utilisant la méthode :

```
1 bool configureTimeStampingPrescaler ( const uint32_t inPrescaler );
```

Le *time base counter* va s'incrémenter chaque **inPrescaler** × **SYSCLK**, avec **inPrescaler** ∈ [1, 1023].

3/Configurer la position d'horodatage en utilisant la méthode :

```
1 bool configureTimeStampingPosition ( const bool atBeginning , const bool  
atSOForRES = 0 );
```

- **atBeginning** : si **atBeginning**=0 alors l'horodatage va se faire à la fin de transmission de la trame, sinon si **atBeginning**=1 alors l'horodatage va se faire au début de la transmission de la trame (si CAN2.0B alors après le SOF, sinon si CAN FD alors voir **atSOForRES**).
- **atSOForRES** : si **atBeginning**=1 et trame CANFD alors l'horodatage va se faire après le bit SOF si **atSOForRES**=0 ou bien après le bit RES si **atSOForRES**=1.

4/Activer l'horodatage en utilisant la méthode :

```
1 bool enableTimeStamping ( void );
```

## 5 Les méthodes d'envoi et de réception

### 5.1 La classe CANFDFrame

la classe CANFDFrame va servir pour garder les informations de la trame (ID, DLC, ...). Le diagramme de la classe est le suivant :

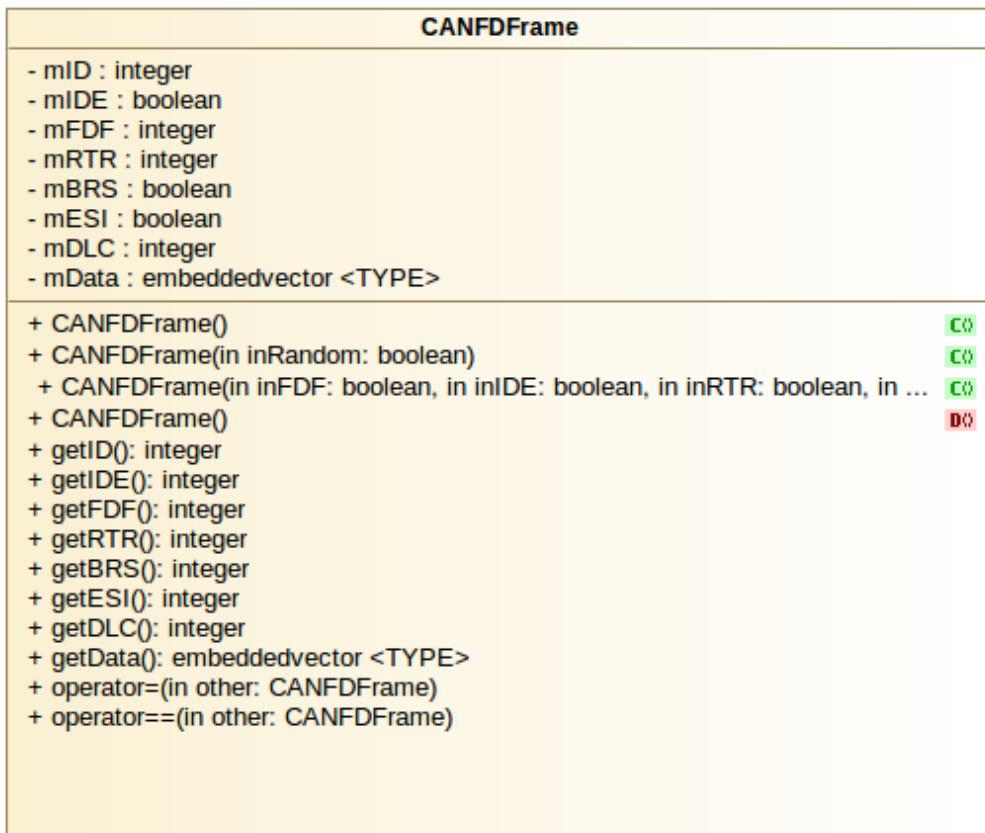


FIGURE 3.6 – Le diagramme de la classe CANFDFrame

### 5.2 Les constructeurs de la classe

- Le constructeur par défaut crée un objet avec les valeurs : mID=0, mIDE=0, mFDF=0, mRTR=0, mBRS=0, mESI=0, mDLC=0, mData=NULL (trame standard d'ID=0x00000 et pas de champs de données).
- Le constructeur **CANFDFrame(bool inRandom)** crée un objet avec des valeurs aux hasard.
- Le constructeur **CANFDFrame(bool inFDF, bool inIDE, bool inRTR, uint32\_t inID, bool inBRS, bool inESI, uint8\_t inDataLength, embeddedvector <uint8\_t> inData)** crée un objet avec les paramètres entrées.

**Remarque :** `embeddedvector <TYPE>` est une classe pour manipuler les tableaux dynamiques (voir Annexe. C, page 55).

### 5.3 Les méthodes d'envoi

Pour charger une trame dans un canal de transmission (TxQ, FIFO), il faut utiliser la méthode suivante :

```
1 bool sendFrame (const uint32_t inChannel, const CANFDFrame &inFrame, const bool  
inDontSend = 0);
```

- **inChannel** : inChannel=0 pour TxQ, inChannel ∈ [1, 31] pour FIFOm.
- **inFrame** : objet CANFDFrame.
- **inDontSend** : si inDontSend=0 alors on va demander la transmission de la trame après l'avoir chargée, sinon si inDontSend=1 alors on va juste charger la trame dans le canal.

Pour demander la transmission des trames dans un canal spécifique ou bien de tous les trames, il faut utiliser la méthode suivante :

```
1 bool requestTransmission (const uint8_t inTransmitMode = 0, const uint32_t  
inChannel = 0);
```

- **inTransmitMode** : si inTransmitMode=0 alors demander la transmission des trames dans tous les canaux, sinon si inTransmitMode=1 alors demander la transmission des messages dans inChannel (0 :TxQ, [1, 31] :FIFOm). Si inTransmitMode>1 alors demander la transmission des messages dans les canaux où inChannel[i]=1 (MSB à droite).
- **inChannel** : inChannel=0 pour TxQ, inChannel ∈ [1, 31] pour FIFOm.

Exemple : requestTransmission(5, 0x00013) => demander la transmission des message dans TxQ (inChannel[31]=1), FIFO1 (inChannel[30]=1) et FIFO4 (inChannel[27]=1).

#### Remarques :

- L'ordre de transmission général dépend de la priorité du canal (inPriority).
- Dans TxQ l'ordre de transmission dépend de l'ID du message.
- Dans les FIFOs l'ordre est : le premier entré, le premier servi.

Pour annuler la transmission dans un canal (inChannel=0 pour TxQ, inChannel ∈ [1, 31] pour FIFOm), il faut utiliser la méthode :

```
1 bool abortTransmission (const uint32_t inChannel);
```

### 5.4 La méthode de réception

Pour recevoir un message d'un canal (inChannel=0 pour TxQ, inChannel ∈ [1, 31] pour FIFOm), il faut utiliser la méthode suivante :

```
1 CANFDFrame* receiveFrame (const uint32_t inChannel);
```

## 6 Test de la bibliothèque en mode LoopBack

Pour tester la bibliothèque on va l'utiliser en mode *InternalLoopbackMode*. On va envoyer des trames au hasard, et afficher le nombre des trames envoyées et reçues sur un afficheur LCD. L'afficheur LCD est contrôlé par un microcontrôleur (PIC 16F690). Le schéma du couplage est le suivant (fait par le logiciel Fritzing[8]) :

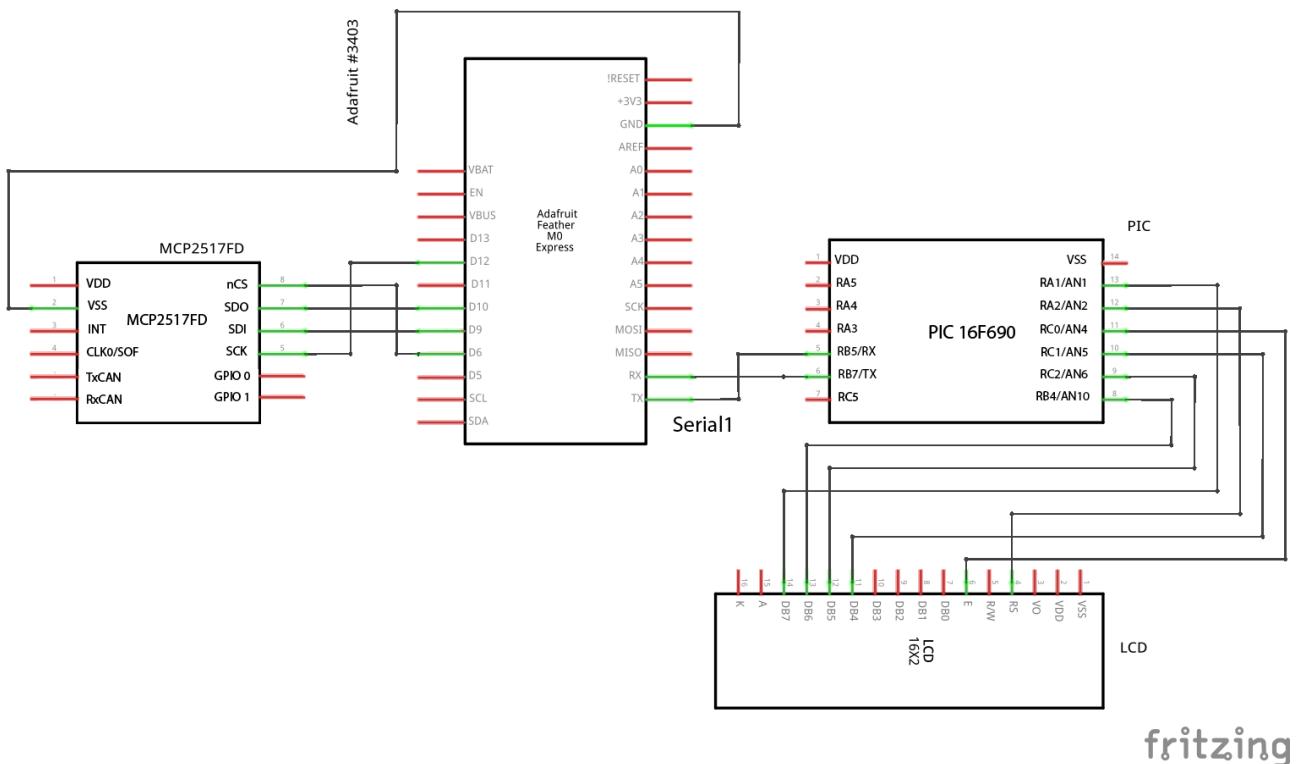


FIGURE 3.7 – Le schéma de couplage en mode InternalLoopback

## 6.1 Configuration du matériel

```
1 #include <Arduino.h>
2 #include <LiquidCrystal.h>
3
4 LiquidCrystal lcd (18, 17, 16, 15, 14, 19);
5
6 void setup()
7 {
8     // initialiser l'afficher LCD      4 lignes et 20 colonnes
9     lcd.begin (20, 4);
10
11    // initialiser Serial1 pour le PIC
12    Serial1.begin (19200);
13 }
```

## 6.2 Configuration du MCP2517FD

On va utiliser le contrôleur aux vitesses : 500kb/s (nominal) et 1Mb/s (data). On va utiliser TxQ pour la transmission et FIFO1 pour la réception.

```
1 #include <wiring_private.h> // pinPeripheral() function
2
3 static const byte MCP2517_CS = 6; // CS input of MCP2517
4 static const byte MCP2517_MISO = 10; // SDO output of MCP2517
5 static const byte MCP2517_MOSI = 11; // SDI input of MCP2517
6 static const byte MCP2517_SCLK = 12; // SCK input of MCP2517
7
8 SPIClass mySPI (&sercom1, MCP2517_MISO, MCP2517_SCLK, MCP2517_MOSI,
9   SPI_PAD_0_SCK_3, SERCOM_RX_PAD_2);
10
11 static const uint32_t nominalBitRate = 500000; // 500 Kb/s
12 static const uint32_t dataBitRate = 1000000; // 1 Mb/s
13 static const uint32_t nominalSamplePoint = 80; //80%
14 static const uint32_t dataSamplePoint = 90; //90%
15 static const uint32_t busLength = 1; //1 m
16 static const uint32_t trcv = 80; //80 ns
17 static const uint32_t tolerance = 30; //30 PPM
18
19 MCP2517FD canController (mySPI, MCP2517_CS, MCP2517FD::QUARTZ_20MHz,
20   nominalBitRate, dataBitRate, nominalSamplePoint, dataSamplePoint, busLength,
21   trcv, tolerance);
22
22 void setup()
23 {
24   //--- Init SPI
25   Serial.begin(9600);
26   mySPI.begin();
27   pinPeripheral(MCP2517_MOSI, PIO_SERCOM);
28   pinPeripheral(MCP2517_SCLK, PIO_SERCOM);
29   pinPeripheral(MCP2517_MISO, PIO_SERCOM);
30
31   //--- Init CAN Controller
32   uint32_t errorCode = canController.begin();
33   if (errorCode == 0)
34   {
35     lcd.setCursor (0, 0); lcd.print ("MCP2517FD ok") ;
36
37     //choose configuration mode
38     canController.configureMode(MCP2517FD::ConfigurationMode);
39
40     //disable TEF;
41     canController.configureTEF(0);
42
43     //enable TXQ, 10 messages deep with 64 bytes data payload
44     canController.configureTXQ(1, 10, 64);
45
46     //set 3 retransmission attempts
47     canController.changeTXQRetransmissionAttempts(1);
48
49     //enable FIFO1, Rx, 10 messages deep with 64 bytes data payload
50     canController.configureFIFOm(1, 0, 32, 8);
51
52     //primary filter 1: save to FIFO1, accept all standard and extended id
53     canController.configureFilterm (1, 1, 1, 0x000, 0x000, 0x00000, 0);
54 }
```

```

52     //choose InternalLoopbackMode
53     canController.configureMode(MCP2517FD::InternalLoopbackMode);
54
55     lcd.setCursor(0, 2); lcd.print("Sent:");
56     lcd.setCursor(0, 3); lcd.print("Received:");
57 }
58 else
59 {
60     lcd.print("MCP2517FD_err_0x");
61     lcd.print(errorCode, HEX);
62 }
63 }
64 }
```

### 6.3 Envoi et réception des trames

on va utiliser le bouton P0 pour envoyer une trame au hasard et le bouton P1 pour envoyer une trame standard CANFD avec changement de vitesse dans la zone data et à deux octets de données.

```

1 #include "MCP2517FD.h"
2 #include "CANFDFrame.h"
3
4 static uint32_t gReceiveDate = 0;
5 static uint32_t gReceivedFrames = 0;
6 static uint32_t gSentFrames = 0;
7
8 CANFDFrame *receivedFrame(NULL), *sentFrame1(NULL);
9
10 void loop()
11 {
12     if (gReceiveDate < millis())
13     {
14         gReceiveDate += 1000;
15         lcd.setCursor(10, 2); lcd.print(gSentFrames);
16         lcd.setCursor(10, 3); lcd.print(gReceivedFrames);
17     }
18
19     //receive frames in FIFO1
20     receivedFrame = canController.receiveFrame(1); //receive frame in FIFO1
21     if(receivedFrame1 != NULL) //frame received
22     {
23         gReceivedFrames++;
24         delete receivedFrame1; receivedFrame1 = NULL;
25     }
26
27     //send a random frame
28     sentFrame1 = new CANFDFrame(1);
29     if(canController.sendFrame(0, *sentFrame1, 0))
30     {
31         gSentFrames++;
32     }
33     delete sentFrame1; sentFrame1 = NULL;
34 }
```

Le test a passé avec succès (réception de toutes les trames envoyées) :



FIGURE 3.8 – Test de la bibliothèque MCP2517FD en mode *InternalLoopback*

# Chapitre 4

## Test de la bibliothèque en mode normal et vérification du programme générateur de trames CAN2.0B et CANFD

Dans ce chapitre, on va tester la bibliothèque en mode normal. On va envoyer des trames générées au hasard par le programme générateur des trames, réalisé précédemment dans le sujet du séminaire bibliographique [9]. Les trames vont être générées sur la carte **Teensy 3.6** et envoyées sur l'émetteur-récepteur CAN **MCP2562**.

# 1 Introduction

On va utiliser le microcontrôleur **Teensy 3.6**, liée à un émetteur-récepteur CAN **MCP2562**; pour envoyer les trames générées par le programme générateur de trames. La trame va être reçu par le contrôleur **MCP2517FD** connecté au microcontrôleur **Adafruit Feather M0**. Le schéma de couplage est le suivant :

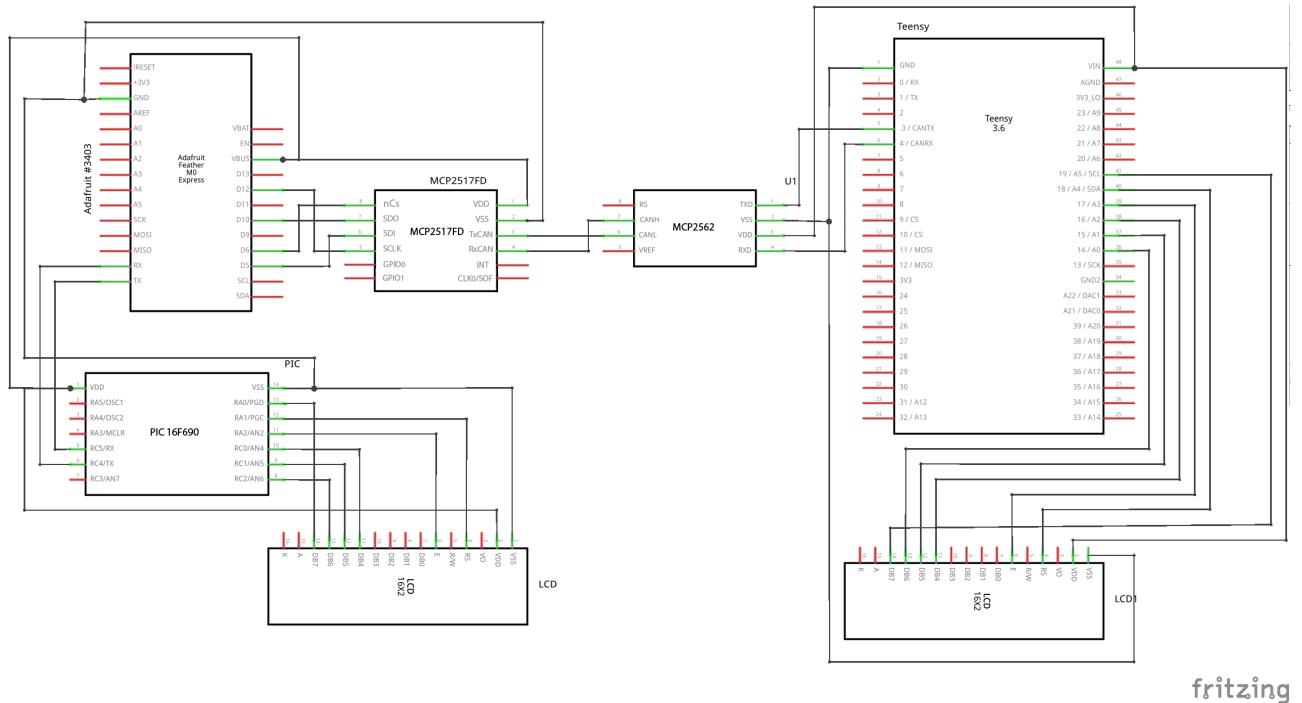


FIGURE 4.1 – Le schéma de couplage en mode Normal

## 2 Configuration du matériel

La fonction “*generate\_frame()*” prend comme paramètres un objet de type *CANFDFrame* et renvoie un tableau de booléens (*embeddedvector <bool>*).

On va utiliser un *Timer* pour envoyer à chaque t ( $t(s) = \frac{1}{bitRate(bits/s)}$ ), un bit de la trame sur le port 29 (PTB18) lié au port TxCAN du **MCP2562**.

Pour modifier PTB18, il faut modifier un bit du mot GPIOB\_PDOR (Adresse = 0x400F\_F040). La méthode usuelle sera de lire le mot complet, mettre le bit à 0 ou à 1 (en faisant un “et logique” et un “ou logique” avec les valeurs correspondantes), et enfin, écrire le mot.

Cette méthode prend du temps et pose des problèmes lorsque l’application doit faire plusieurs choses à la fois (une interruption arrive entre la lecture et l’écriture du mot). Pour éviter ces problèmes, on va utiliser la méthode du “*bit banding*” de ARM (voir Annexe.D, page 58).

```
1 #include <Arduino.h>
2 #include <TimerOne.h>
3 #include "LiquidCrystal.h"
4
5 LiquidCrystal lcd (18, 17, 16, 15, 14, 19);
```

```

6 //bit banding
7 static const uint32_t adresseBitBand = 0x42000000 + (0x000FF040*32) + (18*4);
8 #define BIT_BAND_ADDR (* ((volatile uint32_t *) adresseBitBand))
9
10
11 static uint32_t gBlinkTime = 0;
12 static const uint32_t BIT_RATE = 500 * 1000;
13
14 void setup()
15 {
16     //utilisation d'un ecran 20 colonnes et 4 lignes
17     lcd.begin (20,4);
18     lcd.print ("Sent:");
19
20     //use CAN0 TX alternate pin for output frame
21     digitalWrite (29, HIGH);
22     pinMode (29, OUTPUT);
23
24     //Timer 1 generates periodic interrupt
25     Timer1.initialize (1000000 / BIT_RATE); // Time unit is 1 us
26     Timer1.attachInterrupt (periodicISR);
27 }
28
29 static void periodicISR (void)
30 {
31 }
32
33 void loop()
34 {
35     if (gBlinkTime <= millis ())
36     {
37         gBlinkTime += 1000 ;
38         lcd.setCursor(10, 1)
39         lcd.print(gSent);
40     }
41 }
```

On va configurer le MCP2517FD de la même façon que dans la section 6.2 page 38, la seule modification c'est qu'on va choisir le mode *NormalMode* à la place de *InternalLoopbackMode*.

### 3 Test du programme générateur de trames et du mode normal de la bibliothèque MCP2517FD

On va programmer le programme du *Timer* (periodicISR) pour envoyer chaque t (dans ce cas chaque  $2\mu s$ ), le bit frame[i] (le tableau de la trame générées) et des '1' à la fin de la transmission de la trame.

```

1 #include "embeddedvector.h"
2 #include "CANFDFrame.h"
3 #include "generator_functions.h"
4
5 static embeddedvector<bool> frame; //la trame generee
6 static uint32_t gBitArrayLength = 0; //la taille de la trame
7 static volatile uint32_t gIndex = UINT32_MAX; //l'index pour parcourir les
     elements du tableau
```

```

8 static uint32_t gSent = 0;
9 static bool gEmettre = false; //condition pour generer une trame
10 static uint32_t retransmit = 0; //variable pour marquer la fin de transmission
11   de la trame
12
13 void setup()
14 {
15     gEmettre = true; //commencer generation des trames
16 }
17
18 static void periodicISR (void)
19 {
20     if (gIndex < gBitArrayLength) //envoyer les bits de la trames
21     {
22         BIT_BAND_ADDR = frame[gIndex] ;
23         gIndex ++ ;
24         if(gIndex == gBitArrayLength)
25             retransmit = 0;
26     }
27     else //envoyer des '1'
28     {
29         BIT_BAND_ADDR = 1 ; // RECESSIF bit
30         retransmit++;
31         if( retransmit == 15)
32             gEmettre = true;
33     }
34 }
35
36 void loop()
37 {
38     if (gEmettre)
39     {
40         gEmettre = false ;
41
42         bool fdf = (random () & 1) != 0;
43         bool extended = (random () & 1) != 0;
44         uint32_t identifier = (uint32_t) random ();
45         bool brs = (random () & 1) != 0;
46         bool esi = (random () & 1) != 0;
47         uint8_t dataLength = ((uint8_t) random ()) % 9 ;
48         embeddedvector<uint8_t> data;
49         for (unsigned int i=0 ; i<dataLength ; i++) {
50             data.push((uint8_t) random());
51         }
52
53         CANFDFrame inFrame(fdf, extended, 0/*RTR*/, identifier, brs, esi, dataLength
54           , data);
55         frame = generate_frame(inFrame);
56
57         gBitArrayLength = frame.length();
58         gIndex = 0 ; // Start emitting frame
59         gSent ++ ;
60     }
61 }
```

Le test du mode normal, ainsi que du programme générateur des trames a passé avec succès :

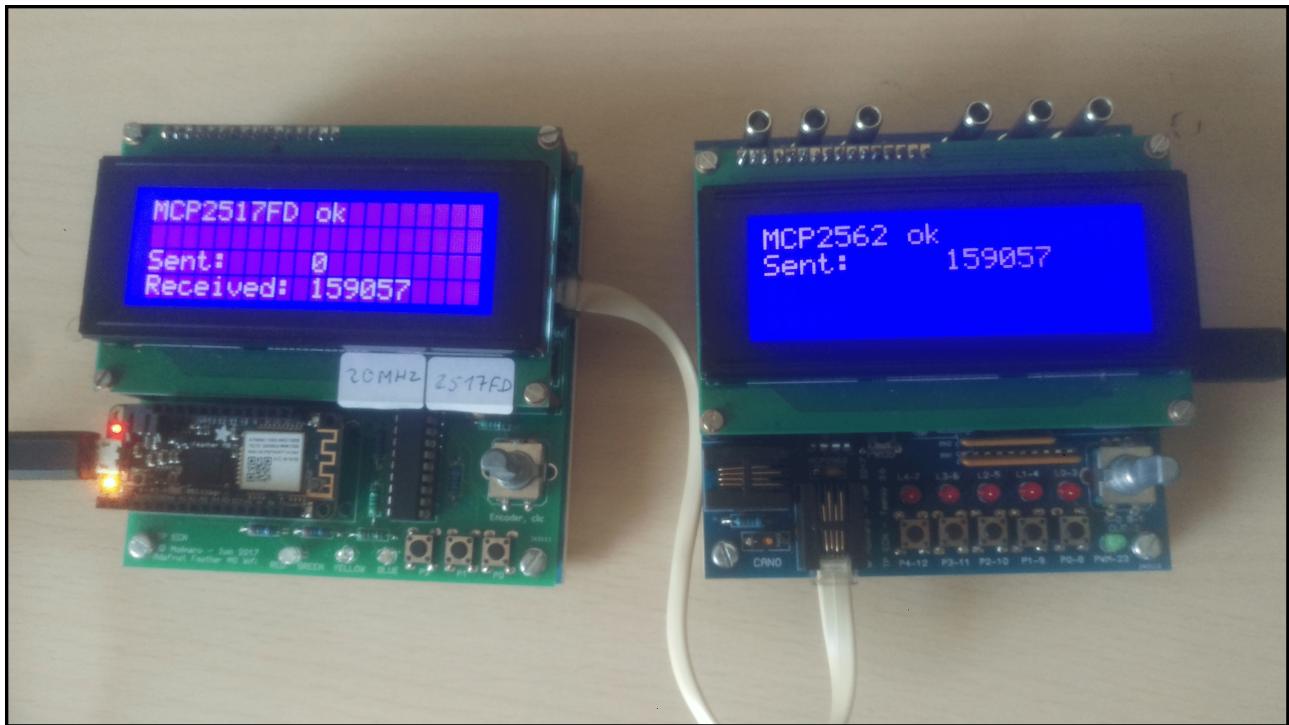


FIGURE 4.2 – Test de la bibliothèque MCP2517FD en mode *Normal*

# Conclusion

L'objectif principal de ce projet est d'une part d'écrire un pilote pour le contrôleur **MCP2517FD**, et d'autre part de vérifier le programme générateur de trames CAN 2.0B et ISO CAN FD.

Dans la première partie, nous avons introduit le réseau CAN et sa nouvelle version CAN FD.

Dans la deuxième partie, nous avons présenté des algorithmes pour configurer le *Nominal bit time* et le *Data bit time*.

Dans la troisième partie, nous avons commencé par la présentation des différents blocs de la bibliothèque **MCP2517FD**, par la suite, nous avons présenté les fonctions du configuration du contrôleur, ensuite, nous avons montré les fonctions d'envoi et de réception des trames, et enfin, nous avons vu un exemple d'utilisation de la bibliothèque en mode *InternalLoopback*.

Dans le dernier chapitre, nous avons testé la bibliothèque en mode de fonctionnement normal, puis, nous avons testé le programme générateur de trames.

Le test de la bibliothèque et du programme générateur de trames a passé avec succès.

# Bibliographie

- [1] P. Richards, *Understanding Microchip's CAN Module Bit Timing*. Microchip Technology Inc, 2001.
- [2] “Road Vehicles - Controller Area Network (CAN)”, 2015.
- [3] S. BOURAGBA, “Etude de la différence entre le non ISO CAN FD et l'ISO CAN FD”, [https://drive.google.com/open?id=1\\_BTVbbGSokDdYB23vTrcG0XhwcG7sgq6](https://drive.google.com/open?id=1_BTVbbGSokDdYB23vTrcG0XhwcG7sgq6), 02 2018.
- [4] “External CAN FD Controller with SPI Interface”, <http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2517FD-External-CAN-FD-Controller-with-SPI-Interface-20005688B.pdf>, 2018.
- [5] “MCP2517FD”, <https://github.com/SalahEddineBOURAGBA/MCP2517FD>, 2018.
- [6] “Modelio Open Source - UML and BPMN free modeling tool”, <https://www.modelio.org/>, 2018.
- [7] lady ada, “Using ATSAMD21 SERCOM for more SPI, I2C and Serial ports”, <https://learn.adafruit.com/using-atsamd21-sercom-to-add-more-spi-i2c-serial-ports/overview>, 04 2017.
- [8] “fritzing - electronics made easy”, <https://www.fritzing.org>, 2018.
- [9] S. BOURAGBA, “CANFD frames generator”, [https://github.com/SalahEddineBOURAGBA/CAN-FD\\_Frame\\_Generator](https://github.com/SalahEddineBOURAGBA/CAN-FD_Frame_Generator), 02 2018.
- [10] P. Molinaro, “Cours - Systèmes interconnectés”, 01 2018
- [11] ARM, *ARM Cortex-M4 Processor*, 2015.

# Annexe A

## Calcul de l'intervalle du PhaseSeg2

Le document “ISO 11898-1 :2015” définit les conditions sur la tolérance d’horloge (paragraphe 11.3.2.5).

D’après l’équation (4) du document on a :  $df < \frac{\min(NominalPhaseSeg1, NominalPhaseSeg2)}{2 \cdot (13 \cdot NominalBTQ - NominalPhaseSeg2)}$

Cas 1 :  $NominalPhaseSeg2 \leq NominalPhaseSeg1$  (si on cherche la valeur minimal du  $NominalPhaseSeg2$ ) :

$$\begin{aligned} df &< \frac{NominalPhaseSeg2}{2 \cdot (13 \cdot NominalBTQ - NominalPhaseSeg2)} \\ 26 \cdot df \cdot NominalBTQ - 2 \cdot df \cdot NominalPhaseSeg2 &< NominalPhaseSeg2 \\ 26 \cdot df \cdot NominalBTQ &< NominalPhaseSeg2 \cdot (1 + 2 \cdot df) \\ NominalPhaseSeg2 &> \frac{26 \cdot NominalBTQ \cdot df}{1 + 2 \cdot df} \end{aligned} \quad (\text{A.1})$$

Cas 2 :  $NominalPhaseSeg2 > NominalPhaseSeg1$  (si on cherche la valeur maximal du  $NominalPhaseSeg2$ ) :

$$\begin{aligned} df &< \frac{NominalPhaseSeg1}{2 \cdot (13 \cdot NominalBTQ - NominalPhaseSeg2)} \\ df &< \frac{NominalBTQ - (1 + NominalPropSeg - NominalPhaseSeg2)}{2 \cdot (13 \cdot NominalBTQ - NominalPhaseSeg2)} \\ 26 \cdot df \cdot NominalBTQ - 2 \cdot df \cdot NominalPhaseSeg2 &< NominalBTQ \\ -(1 + NominalPropSeg - NominalPhaseSeg2) & \\ NominalPhaseSeg2 \cdot (1 - 2 \cdot df) &< (NominalBTQ - (1 + NominalPropSeg)) \\ -26 \cdot df \cdot NominalBTQ \\ NominalPhaseSeg2 &< \frac{(NominalBTQ - (1 + NominalPropSeg)) - 26 \cdot df \cdot NominalBTQ}{(1 - 2 \cdot df)} \end{aligned} \quad (\text{A.2})$$

D'après l'équation (7) du document on a :

$$df < \frac{DataSJW - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1)}{[(2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} + DataPhaseSeg2 + 4 \cdot DataBTQ]}$$

Cas 1 :  $DataPhaseSeg2 < DataPhaseSeg1 \implies DataSJW = DataPhaseSeg2$  (si on cherche la valeur minimal du  $DataPhaseSeg2$ ) :

$$DataPhaseSeg2 - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) > 2 \cdot df \cdot DataPhaseSeg2 + 8 \cdot df \cdot DataBTQ \\ + 2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler}$$

$$DataPhaseSeg2 \cdot (1 - 2 \cdot df) > 8 \cdot df \cdot DataBTQ + 2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2).$$

$$DataPhaseSeg2 > \frac{\frac{NominalPrescaler}{DataPrescaler} - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) + 8 \cdot df \cdot DataBTQ - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1)}{(1 - 2 \cdot df)} \quad (A.3)$$

Cas 2 :  $DataPhaseSeg2 > DataPhaseSeg1 \implies DataSJW = DataPhaseSeg1$  (si on cherche la valeur maximal du  $DataPhaseSeg2$ ) :

$$DataPhaseSeg1 - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) > 2 \cdot df \cdot DataPhaseSeg2 + 8 \cdot df \cdot DataBTQ \\ + 2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler}$$

$$(DataBTQ - 1) - DataPhaseSeg2 - \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) > 2 \cdot df \cdot DataPhaseSeg2 \\ + 8 \cdot df \cdot DataBTQ + 2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler}$$

$$DataPhaseSeg2 \cdot (1 + 2 \cdot df) < DataBTQ - \left[ 1 + 8 \cdot df \cdot DataBTQ + 2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} + \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) \right]$$

$$DataPhaseSeg2 < \frac{DataBTQ - \left[ 1 + 8 \cdot df \cdot DataBTQ + \max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) + 2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} \right]}{1 + 2 \cdot df} \quad (A.4)$$

## Annexe B

# Vérification de la portée des variables (range check)

Pour assurer le bon fonctionnement du programme, on doit vérifier que à tous moment, toute variable est dans ses limites de valeurs.

Pour des raisons d'optimisation, on ne va travailler que par les entiers 'Integers' et éviter l'utilisation des réels 'Floaters'. En effet, l'utilisation du type flottant augmente le temps d'exécution du programme par 4. [10]

Carte	Feather M0 Wifi
100 000 additions + divisions entiers 32 bits	382 ms
100 000 additions + divisions flottants 32 bits	1488 ms

TABLE B.1 – Le temps d'exécution des opérations en entier et en flottant

## 1 Configuration du *bitTime*

Toutes les variables sont des entiers non signés de 32 bits (**uint32\_t**), ils varient donc dans l'intervalle  $[0; 4.29 \cdot 10^9]$ .

- $NominalPrescaler = \frac{f_{CANFD}}{NominalBTQ \cdot NominalBitRate}$

$$\begin{aligned} \text{On a : } 4 \cdot 10^6 &\leq f_{CANFD} \leq 40 \cdot 10^6 \implies \frac{4 \cdot 10^6}{32 \cdot 80} \leq NominalBitRate \leq \frac{40 \cdot 10^6}{1 \cdot 8} \\ &\implies 1562 \leq NominalBitRate \leq 5 \cdot 10^6 \end{aligned}$$

$$\begin{aligned} \text{et : } 8 &\leq NominalBTQ \leq 80 \implies 12496 \leq NominalBTQ \cdot NominalBitRate \leq 4 \cdot 10^8 \checkmark \\ &\implies \frac{4 \cdot 10^6}{4 \cdot 10^8} \leq \frac{f_{CANFD}}{NominalBTQ \cdot NominalBitRate} \leq \frac{40 \cdot 10^6}{12496} \\ &\implies 0 \leq \frac{f_{CANFD}}{NominalBTQ \cdot NominalBitRate} \leq 3201 \checkmark \end{aligned}$$

- $NominalPropSeg = \frac{t_{prop}(ns)}{10^9} \cdot \frac{f_{CANFD}}{NominalPrescaler}$
- On a :  $3 \leq \delta(ns/m) \leq 10$        $\Rightarrow 0 \leq 2(\delta \cdot L + trcv) \leq 4 \cdot 10^4$   
 $0 < L(m) \leq 1\,000$        $\Rightarrow 0 \leq t_{prop}(ns) \leq 4 \cdot 10^4$   
 $0 \leq trcv(ns) \leq 10\,000$

$$\text{et on a : } 4 \cdot 10^6 \leq f_{CANFD} \leq 40 \cdot 10^6 \Rightarrow 4 \leq \frac{f_{CANFD}}{10^6} \leq 40$$

$$\Rightarrow 0 \leq t_{prop} \cdot \frac{f_{CANFD}}{10^6} \leq 16 \cdot 10^5 \checkmark$$

$$1 \leq NominalPrescaler \leq 25 \Rightarrow 10^3 \leq 10^3 \cdot NominalPrescaler \leq 25 \cdot 10^3 \checkmark$$

$$\text{donc : } 0 \leq \frac{t_{prop} \cdot \frac{f_{CANFD}}{10^6}}{10^3 \cdot NominalPrescaler} \leq 1\,600 \checkmark$$

- $NominalSJW_{min} = \lceil 20 \cdot NominalBTQ \cdot df \rceil$
- On a :  $8 \leq NominalBTQ \leq 80$    et :  $0 \leq Tolerance \leq 100$   
 $\Rightarrow 0 \leq 20 \cdot NominalBTQ \cdot Tolerance \leq 16 \cdot 10^4 \checkmark$   
 $\Rightarrow 0 \leq \frac{20 \cdot NominalBTQ \cdot Tolerance}{10^6} \leq 0.16$   
 $\Rightarrow 0 \leq NominalSJW_{min} \leq 1 \checkmark$

- $cond4max = \left\lfloor \frac{(NominalBTQ - (1 + NominalPropSeg)) - 26 \cdot df \cdot NominalBTQ}{(1 - 2 \cdot df)} \right\rfloor$

On a :  $8 \leq NominalBTQ \leq 80$    et :  $2 \leq 1 + NominalPropSeg \leq 49$   
 $\Rightarrow -41 \leq NominalBTQ - (1 + NominalPropSeg) \leq 78$

Or la condition est valable dans le cas où :  $NominalBTQ - (1 + NominalPropSeg) \geq 3$

$$\Rightarrow 3 \leq NominalBTQ - (1 + NominalPropSeg) \leq 78$$

$$\Rightarrow 3 \cdot 10^6 \leq 10^6 \cdot (NominalBTQ - (1 + NominalPropSeg)) \leq 78 \cdot 10^6 \checkmark$$

Et on a :  $0 \leq 26 \cdot NominalBTQ \cdot Tolerance \leq 208\,000 \checkmark$

Donc :  $2.79 \cdot 10^6 \leq [10^6 \cdot (NominalBTQ - (1 + NominalPropSeg))] - [26 \cdot NominalBTQ \cdot Tolerance] \leq 78 \cdot 10^6 \checkmark$

Or :  $10^6 - 2 \cdot Tolerance \sim 10^6 \checkmark$

Donc :  $2 \leq \left\lfloor \frac{[10^6 \cdot (NominalBTQ - (1 + NominalPropSeg))] - [26 \cdot NominalBTQ \cdot Tolerance]}{10^6 - 2 \cdot Tolerance} \right\rfloor \leq 78 \checkmark$

- $cond4min = \left\lceil \frac{26 \cdot df \cdot NominalBTQ}{1 + 2 \cdot df} \right\rceil$

On a :  $0 \leq 26 \cdot NominalBTQ \cdot Tolerance \leq 208000$

et :  $10^6 + 2 \cdot Tolerance \sim 10^6$

$$\begin{aligned} \text{donc : } 0 &\leq \frac{26 \cdot Tolerance \cdot NominalBTQ}{10^6 + 2 \cdot Tolerance} \leq 0.0208 \\ \Rightarrow 0 &\leq \left\lceil \frac{26 \cdot Tolerance \cdot NominalBTQ}{10^6 + 2 \cdot Tolerance} \right\rceil \leq 1 \checkmark \end{aligned}$$

- $NominalDiff = \frac{|f_{CANFD} - NominalBitRate \cdot NominalBTQ \cdot NominalPrescaler|}{f_{CANFD}} \cdot 10^6$

On a :  $1562 \leq NominalBitRate \leq 5 \cdot 10^6$  ,  $8 \leq NominalBTQ \leq 80$  ,  
 $1 \leq NominalPrescaler \leq 32$  ,  $4 \cdot 10^6 \leq f_{CANFD} \leq 40 \cdot 10^6$

$\Rightarrow 12496 \leq NominalBitRate \cdot NominalBTQ \cdot NominalPrescaler \leq 1.28 \cdot 10^9 \checkmark$

$\Rightarrow 0 \leq |f_{CANFD} - NominalBitRate \cdot NominalBTQ \cdot NominalPrescaler| \leq 1.276 \cdot 10^9 \checkmark$

et :  $4 \leq \frac{f_{CANFD}}{10^6} \leq 40 \checkmark$

$\Rightarrow 0 \leq \frac{|f_{CANFD} - NominalBitRate \cdot NominalBTQ \cdot NominalPrescaler|}{\frac{f_{CANFD}}{10^6}} \leq 3.19 \cdot 10^8 \checkmark$

- $DataPrescaler = \frac{f_{CANFD}}{DataBTQ \cdot DataBitRate}$

On a :  $4 \cdot 10^6 \leq f_{CANFD} \leq 40 \cdot 10^6 \Rightarrow \frac{4 \cdot 10^6}{25 \times 32} \leq DataBitRate \leq \frac{40 \cdot 10^6}{5 \cdot 1}$   
 $\Rightarrow 5000 \leq DataBitRate \leq 8 \cdot 10^6$

et :  $5 \leq DataBTQ \leq 25 \Rightarrow 25000 \leq DataBTQ \cdot DataBitRate \leq 2 \cdot 10^8 \checkmark$

$$\Rightarrow 0 < \frac{f_{CANFD}}{DataBTQ \cdot DataBitRate} \leq 1600 \checkmark$$

- $DataSJW_{min} = \lceil 20 \cdot DataBTQ \cdot df \rceil$

On a :  $5 \leq DataBTQ \leq 25$  et :  $0 \leq Tolerance \leq 100$

$$\Rightarrow 0 \leq 20 \cdot DataBTQ \cdot Tolerance \leq 5 \cdot 10^4 \checkmark$$

$$\Rightarrow 0 \leq \frac{20 \cdot DataBTQ \cdot Tolerance}{10^6} \leq 0.05$$

$$\Rightarrow 0 \leq DataSJW_{min} \leq 1 \checkmark$$

$$\bullet \ cond7max = \left[ \frac{DataBTQ - \left[ 1 + 8 \cdot df \cdot DataBTQ + max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) + 2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} \right]}{1 + 2 \cdot df} \right]$$

On a :  $1 \leq NominalPrescaler \leq 32$ , et :  $1 \leq DataPrescaler \leq 32$

$$\begin{aligned} \Rightarrow 0 \leq \frac{NominalPrescaler}{DataPrescaler} \leq 32 &\Rightarrow 0 \leq max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) \leq 31 \\ \Rightarrow 10^6 \leq 10^6 \cdot (1 + max(0, \frac{NominalPrescaler}{DataPrescaler} - 1)) &\leq 32 \cdot 10^6 \checkmark \end{aligned}$$

et on a :  $0 \leq 8 \cdot Tolerance \cdot DataBTQ \leq 2 \cdot 10^4 \checkmark$

On a :  $2 \leq NominalPhaseSeg2 \leq 16$  et  $16 \leq 2 \cdot NominalBTQ \leq 160$

$$\begin{aligned} \Rightarrow 0 \leq 2 \cdot NominalBTQ - NominalPhaseSeg2 &\leq 158 \checkmark \\ \Rightarrow 0 \leq 2 \cdot Tolerance \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) &\leq 3.16 \cdot 10^4 \checkmark \end{aligned}$$

$$\begin{aligned} \Rightarrow 0 \leq 2 \cdot Tolerance \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \\ \cdot \frac{NominalPrescaler}{DataPrescaler} &\leq 1.01 \cdot 10^6 \checkmark \end{aligned}$$

$$\begin{aligned} \text{Soit } \mathbf{A} &= \left( 10^6 \cdot (1 + max(0, \frac{NominalPrescaler}{DataPrescaler} - 1)) \right) + \left( 8 \cdot Tolerance \cdot DataBTQ \right) \\ &+ \left( 2 \cdot Tolerance \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} \right) \end{aligned}$$

On a donc :  $10^6 \leq \mathbf{A} \leq 33 \cdot 10^6 \checkmark$

Si  $10^6 \cdot DataBTQ \leq \mathbf{A}$  alors  $cond7max = 0$

Sinon, on a :  $5 \cdot 10^6 \leq 10^6 \cdot DataBTQ \leq 25 \cdot 10^6 \Rightarrow 0 \leq 10^6 \cdot DataBTQ - \mathbf{A} \leq 15 \cdot 10^6 \checkmark$

Or :  $10^6 + 2 \cdot Tolerance \sim 10^6$

$$\text{Donc : } 0 \leq \frac{10^6 \cdot DataBTQ - \mathbf{A}}{10^6 + 2 \cdot Tolerance} \leq 15 \Rightarrow 0 \leq cond7max \leq 15 \checkmark$$

$$\bullet \ cond7min = \left\lfloor \frac{2 \cdot df \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} + 8 \cdot df \cdot DataBTQ - max(0, \frac{NominalPrescaler}{DataPrescaler} - 1)}{(1 - 2 \cdot df)} \right\rfloor$$

On a :  $0 \leq max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) \leq 31$

$$\Rightarrow 0 \leq 10^6 \cdot max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) \leq 31 \cdot 10^6 \checkmark$$

et on a :  $0 \leq 8 \cdot Tolerance \cdot DataBTQ \leq 2 \cdot 10^4 \checkmark$

$$\begin{aligned} \text{et : } 0 &\leq 2 \cdot Tolerance \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \\ &\quad \cdot \frac{NominalPrescaler}{DataPrescaler} \leq 1.01 \cdot 10^6 \checkmark \end{aligned}$$

Si  $\left[ (2 \cdot Tolerance \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler}) + (8 \cdot Tolerance \cdot DataBTQ) \right] < \left[ 10^6 \cdot max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) \right]$

alors **cond7min = 0**  $\checkmark$

$$\begin{aligned} \text{Sinon : } 0 &\leq \left( 2 \cdot Tolerance \cdot (2 \cdot NominalBTQ - NominalPhaseSeg2) \cdot \frac{NominalPrescaler}{DataPrescaler} \right) \\ &\quad + \left( 8 \cdot Tolerance \cdot DataBTQ \right) - \left( 10^6 \cdot max(0, \frac{NominalPrescaler}{DataPrescaler} - 1) \right) \leq 1.1 \cdot 10^6 \checkmark \end{aligned}$$

Or :  $10^6 - 2 \cdot Tolerance \sim 10^6 \Rightarrow 0 \leq \text{cond7min} \leq 1 \checkmark$

$$\bullet \ DataDiff = \frac{|f_{CANFD} - DataBitRate \cdot DataBTQ \cdot DataPrescaler|}{f_{CANFD}} \cdot 10^6$$

On a :  $5000 \leq DataBitRate \leq 8 \cdot 10^6 \quad , \quad 5 \leq DataBTQ \leq 25 \quad ,$

$$1 \leq DataPrescaler \leq 32 \quad , \quad 4 \cdot 10^3 \leq \frac{f_{CANFD}}{10^3} \leq 40 \cdot 10^3$$

$$\Rightarrow 25 \leq \frac{DataBitRate}{10^3} \cdot DataBTQ \cdot DataPrescaler \leq 6.4 \cdot 10^6 \checkmark$$

$$\Rightarrow 0 \leq \left| \frac{f_{CANFD}}{10^3} - \frac{DataBitRate}{10^3} \cdot DataBTQ \cdot DataPrescaler \right| \leq 1600 \checkmark$$

Or :  $4 \leq \frac{f_{CANFD}}{10^6} \leq 40 \checkmark$

$$\Rightarrow 0 \leq \frac{\left| \frac{f_{CANFD}}{10^3} - \frac{DataBitRate}{10^3} \cdot DataBTQ \cdot DataPrescaler \right|}{\frac{f_{CANFD}}{10^6}} \leq 400 \checkmark$$

$$\Rightarrow 0 \leq \frac{\left| \frac{f_{CANFD}}{10^3} - \frac{DataBitRate}{10^3} \cdot DataBTQ \cdot DataPrescaler \right|}{\frac{f_{CANFD}}{10^6}} \cdot 10^3 \leq 4 \cdot 10^5 \checkmark$$

# Annexe C

## La classe `embeddedvector <TYPE>`

On va utiliser la classe `embeddedvector <TYPE>` pour les tableaux dynamiques, et principalement pour le champs `mData` de la classe `CANFDFrame`, puisque sa taille est dynamique et peut atteindre jusqu'à 64 bytes.

Le diagramme de la classe est le suivant :

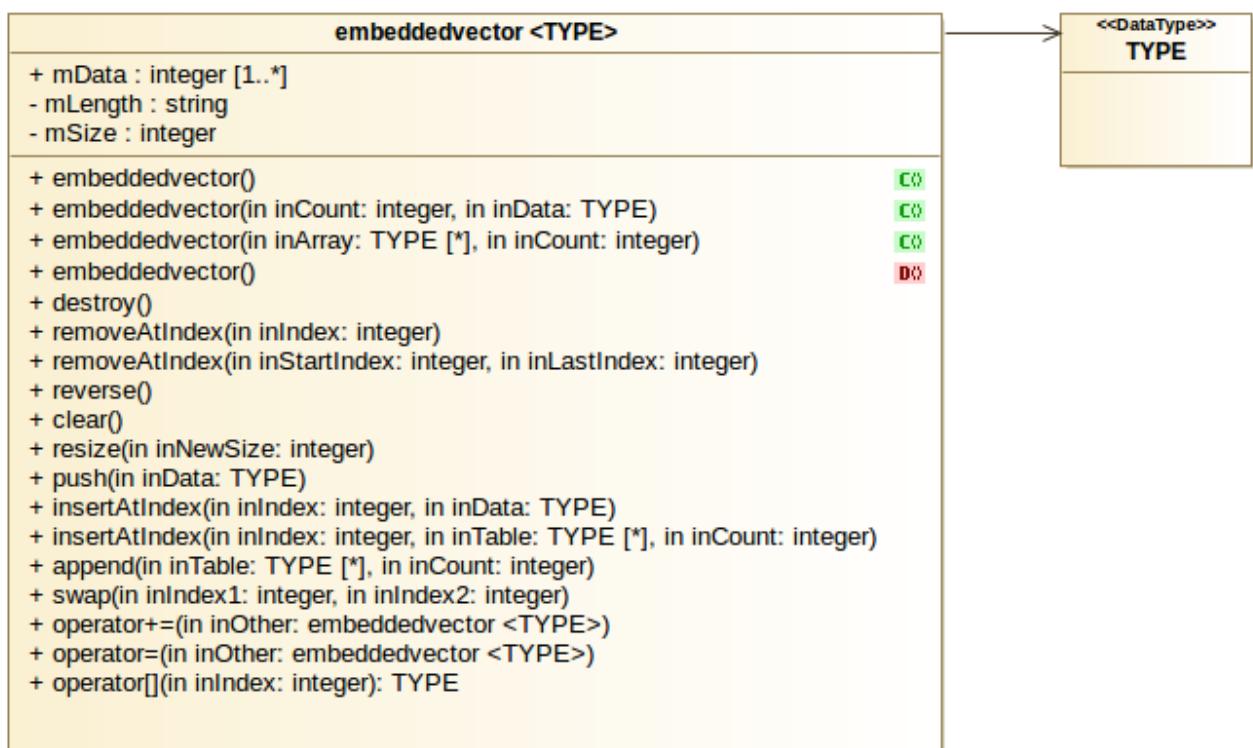


FIGURE C.1 – Le diagramme de la classe `embeddedvector <TYPE>`

### 1 Les attributs de la classe

- **mData** : c'est un pointeur vers un tableau statique de type 'TYPE' et de taille `mLength`.
- **mLength** : la taille d'usage du tableau.

- **mSize** : la taille réelle du tableau.

**Remarque :** La taille réelle du tableau est *mSize*, mais la taille d'usage est *mLength* tq *mSize* = *mLength* + la taille provisoire.

## 2 Les constructeurs de la classe

Le constructeur par défaut crée un tableau de taille réel *mSize* = 20 :

```
1 embeddedvector (void);
```

Le constructeur suivant prend comme paramètres la taille d'usage du tableau et la valeur d'initialisation, il crée un tableau de taille réel *mSize* = 20 + *inCount* :

```
1 embeddedvector (const size_t inCount, const TYPE inData);
```

Le constructeur suivant prend comme paramètres un tableau statique et sa taille, il crée un tableau de taille réel *mSize* = 20 + *inCount* :

```
1 embeddedvector (const TYPE inArray [], const size_t inCount);
```

## 3 Les méthodes de la classe

Pour savoir la taille du tableau, il faut utiliser la méthode suivante :

```
1 size_t length (void);
```

### Ajouter des éléments au tableau

Pour ajouter un seul élément à la fin du tableau, il faut utiliser la méthode suivante :

```
1 void push (const TYPE inData);
```

Pour ajouter un élément à un endroit précis, il faut utiliser la méthode suivante :

```
1 void insertAtIndex (const size_t inIndex, const TYPE inData);
```

Pour ajouter un tableau d'éléments à un endroit précis, il faut utiliser la méthode suivante :

```
1 void insertAtIndex (const size_t inIndex, const TYPE inTable [], const size_t inCount);
```

Pour ajouter un tableau dynamique à la fin du tableau, il faut utiliser l'opérateur '+=' :

```
1 embeddedvector & operator += ( embeddedvector <TYPE> const & inOther );
```

## Retirer des éléments du tableau

Pour supprimer un élément d'un endroit précis, il faut utiliser la méthode suivante :

```
1 void removeAtIndex (const size_t inIndex);
```

Pour supprimer un ensemble d'éléments suivis entre deux positions, il faut utiliser la méthode suivante :

```
1 void removeAtIndex (const size_t instartIndex, const size_t inlastIndex);
```

Pour supprimer tous les éléments du tableau, il faut utiliser la méthode suivante :

```
1 void clear (void);
```

Pour détruire le tableau (libérer l'espace mémoire réservé), il faut utiliser la méthode suivante :

```
1 void destroy (void);
```

# Annexe D

## Le bit banding

Le bit banding a été introduit par **ARM** dans les deux processeurs **Cortex M3 et M4**, afin de faciliter la modification d'un bit dans un mot. La méthode traditionnelle (lire, modifier, écrire) prend 4 instructions mémoires, et est vulnérable à l'interruption (donc peut altérer le résultat).

Le principe c'est qu'on trouve une zone mémoire (*alias region*) où chaque mot est liée à un bit de la zone mémoire (*bit-band region*), la modification d'un mot de la zone '*alias region*' permet de modifier le bit équivalent dans la zone '*bit-band region*' en utilisant une seule instruction.

Dans le **Cortex M4** il existe deux zones '*alias region*' de taille 32Mb qui correspondent chacune à une zone '*bit-band region*' de taille 1Mb. La relation entre les deux est donnée par l'équation suivante :

$$\text{adresse\_mot\_alias} = \text{base\_region\_alias} + (\text{position\_bit} \times 4) + (\text{position\_mot\_du\_bit\_band} \times 32)$$

- **adresse\_mot\_alias** : l'adresse du mot souhaité lié au bit à modifier.
- **base\_region\_alias** : l'adresse du mot de base de la zone '*alias region*'.
- **position\_bit** : la position du bit à modifier dans le mot où il se trouve.
- **position\_mot\_du\_bit\_bit-band** : la position du mot qui contient le bit à modifier dans la zone '*bit-band region*'.

**Exemple :** la figure suivante montre un exemple du *bit banding* entre la zone '*alias region*' et la zone '*bit-band region*' [11] :

- L'adresse du mot lié au bit '0x20000000.0' est :  $0x22000000 = 0x22000000 + (0 \times 4) + (0 \times 32)$ .
- L'adresse du mot lié au bit '0x20000000.7' est :  $0x2200001C = 0x22000000 + (7 \times 4) + (0 \times 32)$ .
- L'adresse du mot lié au bit '0x200FFFFF.0' est :  $0x23FFFFFFE0 = 0x22000000 + (0 \times 4) + (0xFFFF \times 32)$ .
- L'adresse du mot lié au bit '0x200FFFFF.7' est :  $0x23FFFFFFFC = 0x22000000 + (7 \times 4) + (0xFFFF \times 32)$ .

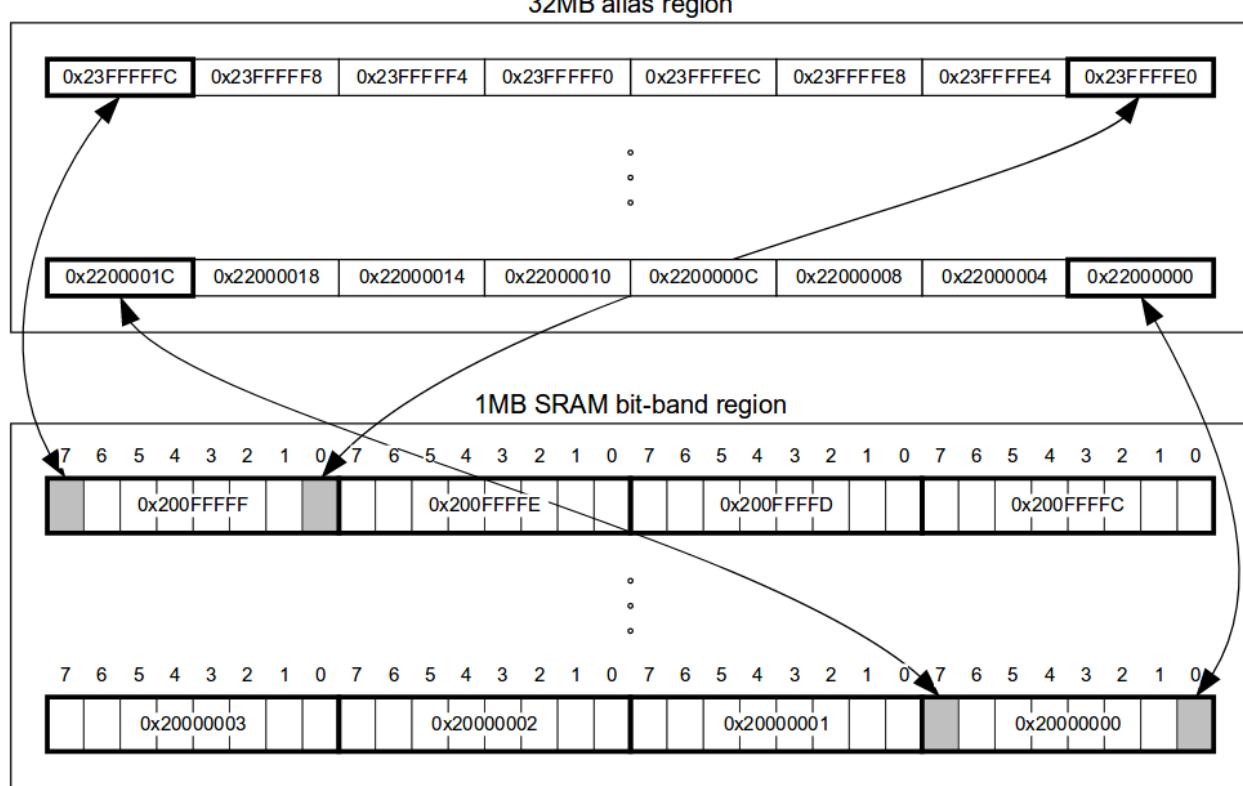


FIGURE D.1 – Le bit banding dans le **Cortex M4**

## Annexe E

# Les codes d'erreur de la configuration du bit time

Dans les algorithmes 2.2, 2.3, 2.5 et 2.6 la variable *ErrorCode* est utilisée chaque fois pour stocker un type d'erreur dans la configuration du *bit time*. La liste des erreurs est :

- ErrorCode.00** : La configuration du *NominalBitTime* ne vérifie pas la tolérance d'horloge.
- ErrorCode.01** : La configuration du *DataBitTime* ne vérifie pas la tolérance d'horloge.
- ErrorCode.02** : Le *NominalSJW* est petit, on ne peut pas assurer la synchronisation à 100%.
- ErrorCode.03** : Le *DataSJW* est petit, on ne peut pas assurer la synchronisation à 100%.
- ErrorCode.04** : Le *NominalPropagationSegment* est trop grand, on ne peut pas assurer le temps de propagation nécessaire dans la phase d'arbitrage.
- ErrorCode.05** : Non utilisé.
- ErrorCode.06** : Le *NominalSamplePoint* voulu est hors limites.
- ErrorCode.07** : Le *DataSamplePoint* voulu est hors limites.
- ErrorCode.08** : Erreur de vitesse du SPI à 1MHz.
- ErrorCode.09** : Erreur de vitesse du SPI à la vitesse demandée.