|  |
| --- |
| Hes-so Valais/Wallis - HEI |
| WP2 : Programmation Steering Module |
| Summer school II - eKart |

|  |
| --- |
| Maxime Caloz  05/09/2022 |

Table des matières

[1 Introduction 2](#_Toc113527140)

[2 Configuration MCC 2](#_Toc113527141)

[2.1 Périphérique ECAN 2](#_Toc113527142)

[2.2 Périphérique MEMORY 2](#_Toc113527143)

[2.3 Périphérique TMR0 2](#_Toc113527144)

[2.4 Périphérique UART1 2](#_Toc113527145)

[2.5 System Module 3](#_Toc113527146)

[3 Schémas UML du code 4](#_Toc113527147)

[4 Phase de tests 8](#_Toc113527148)

[5 Améliorations et conclusion 10](#_Toc113527149)

[6 Table des illustrations 10](#_Toc113527150)

[7 Annexes 10](#_Toc113527151)

# Introduction

Ce document décrit l’implémentation du *module steering* dans le cadre du projet de Summer School 2, le *eKart*.

# Configuration MCC

La configuration *MCC* du module steering nécessite l’implémentation des périphériques suivants :

* ECAN
* MEMORY
* TMR0
* UART1

Leurs configurations sont présentées ci-dessous.

## Périphérique ECAN

La vitesse du bus doit être configurée à 250kBds. Nous utilisons les pins RB3 et RB2 pour respectivement RX et TX0. L’implémentation des filtres nous a été fournie par les professeurs et se trouve dans le fichier *ecan.c*. Nous filtrons sur la destination soit broadcast soit notre identifiant.

## Périphérique MEMORY

Il n’y a pas de configuration spécifique à faire pour ce périphérique.

## Périphérique TMR0

Comme le montre la figure suivante (Figure 1), nous utilisons le timer 0 pour avoir une interruption toutes les 10ms.

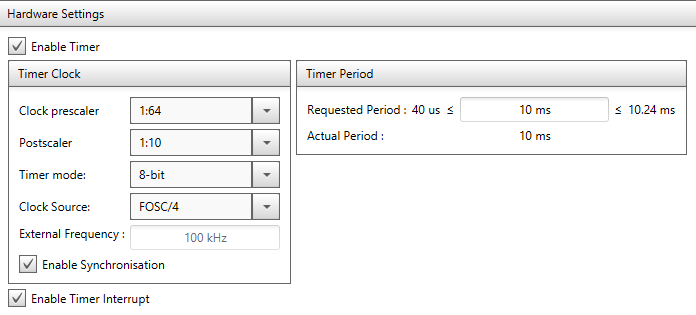


Figure : Configuration du périphérique TMR0

## Périphérique UART1

Ce périphérique est utilisé pour la communication entre le PIC et le driver du monteur d’entraînement. La vitesse du bus doit être de 115200Bds. Nous utilisons les pins RC7 et RC4 pour respectivement RX1 et TX1.

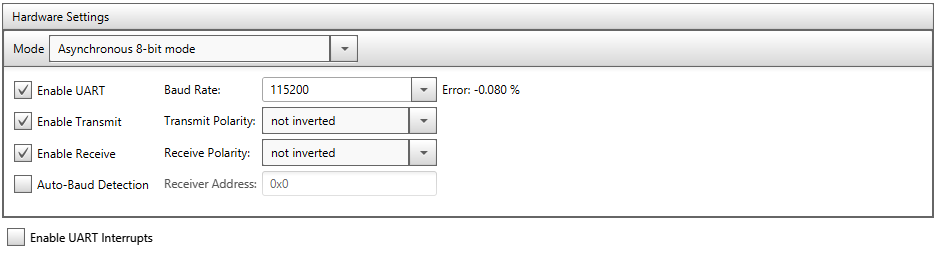


Figure : Configuration du périphérique UART1

## System Module

Les figures suivantes présentent la configuration du système général. Nous avons rajouté un WatchDog afin de reset le PIC dès qu’il reste bloqué plus d’une seconde.

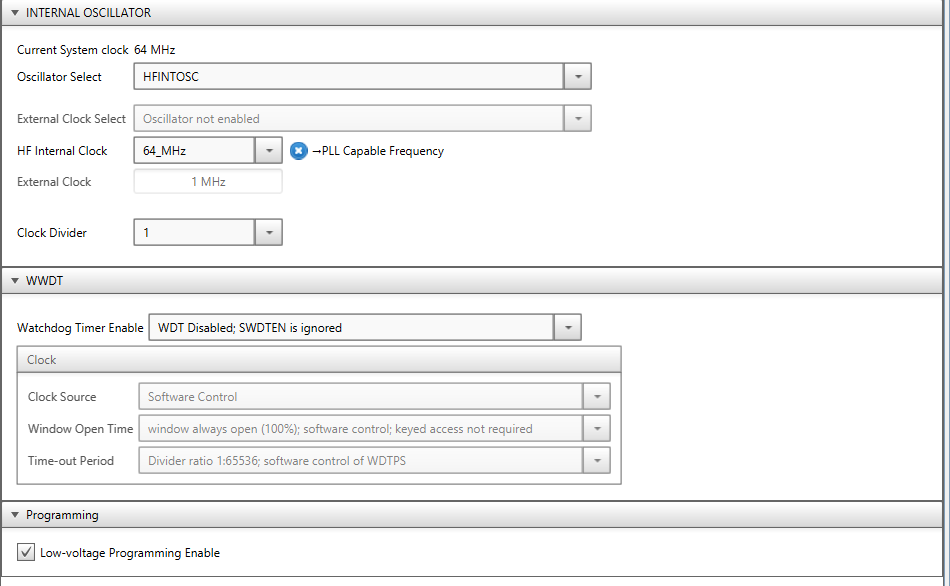


Figure : Configuration du System Module

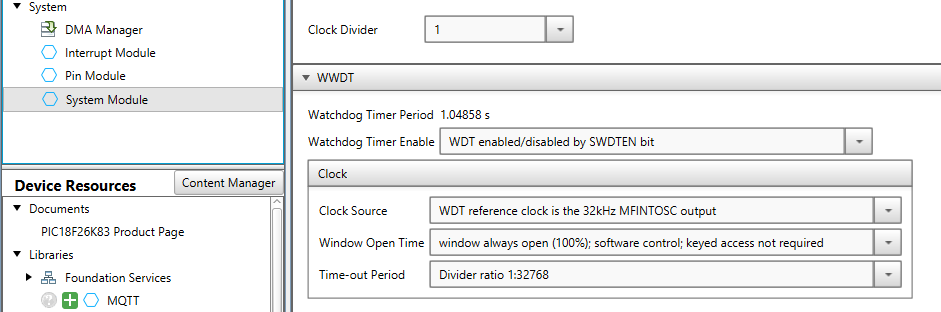


Figure : Configuration du System Module - WatchDog

# Schémas UML du code

Ci-dessous le diagramme de classe de notre programme (Figure 5). Comme nous pouvons le remarquer, notre programme est en C orienté objet. Pour ce faire, nous utilisons la classe *Factory* qui crée tous les objets et possède des getters pour chacun d’eux. Dans le reste du programme, nous utilisons ces derniers pour accéder aux différents objets.

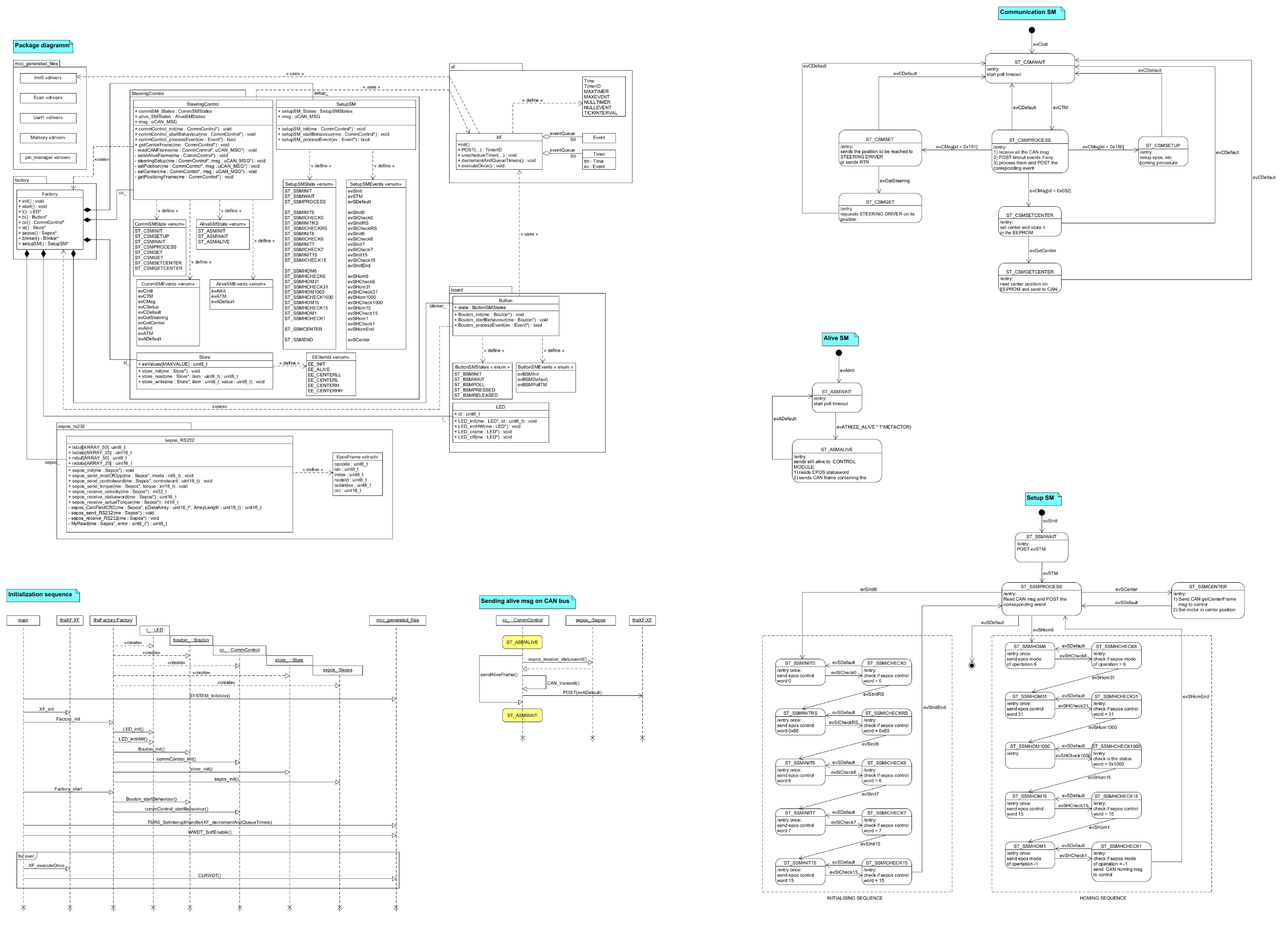


Figure Diagramme de classe

Notre code repose sur trois machines d’état. La première gère la réception des messages CAN (Figure 8), la deuxième la gestion du setup du drive Epos (Figure 6) et la dernière l’envoi du keep alive (Figure 7).

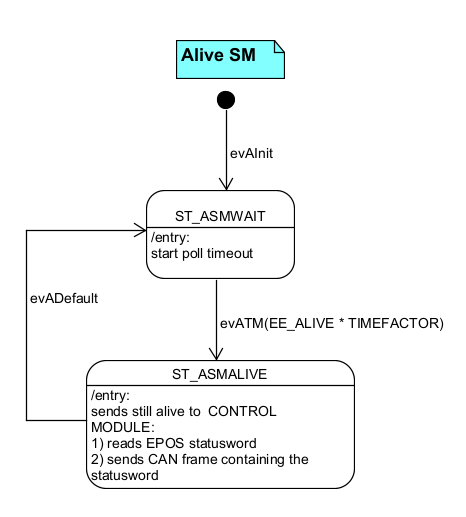


Figure 6 State Machine Alive

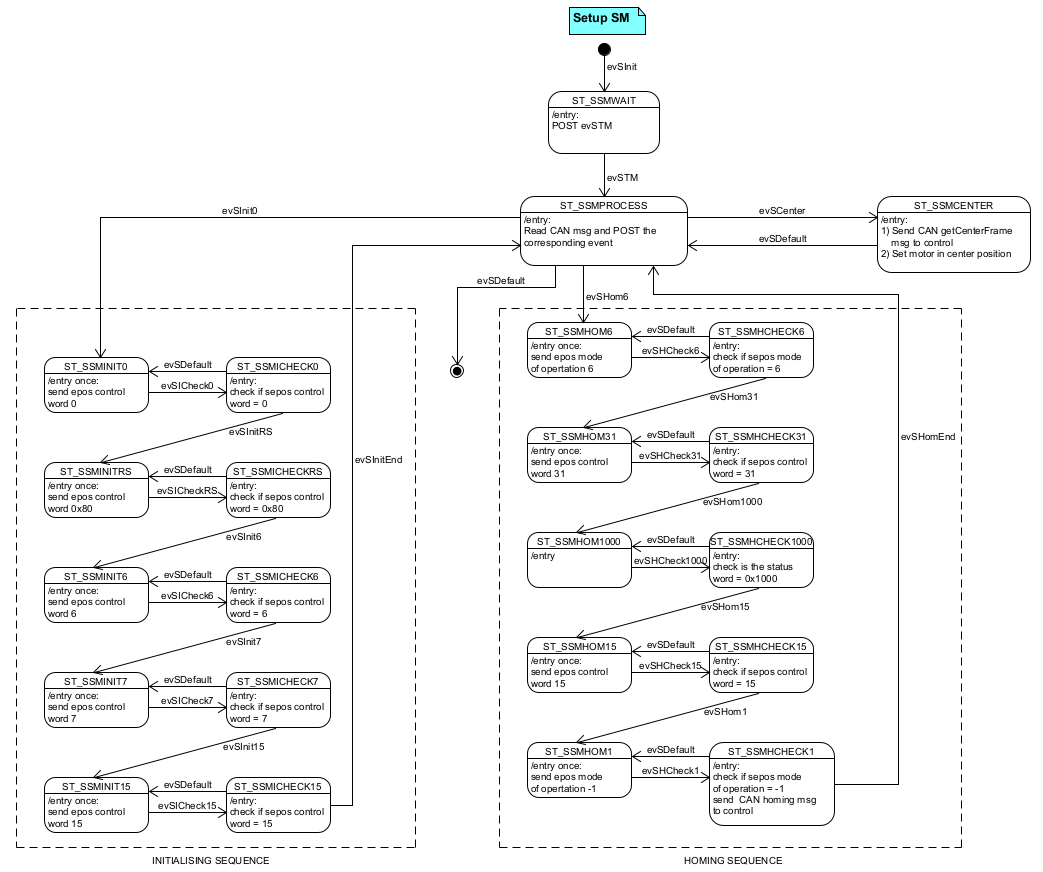


Figure 8 State Machine Setup

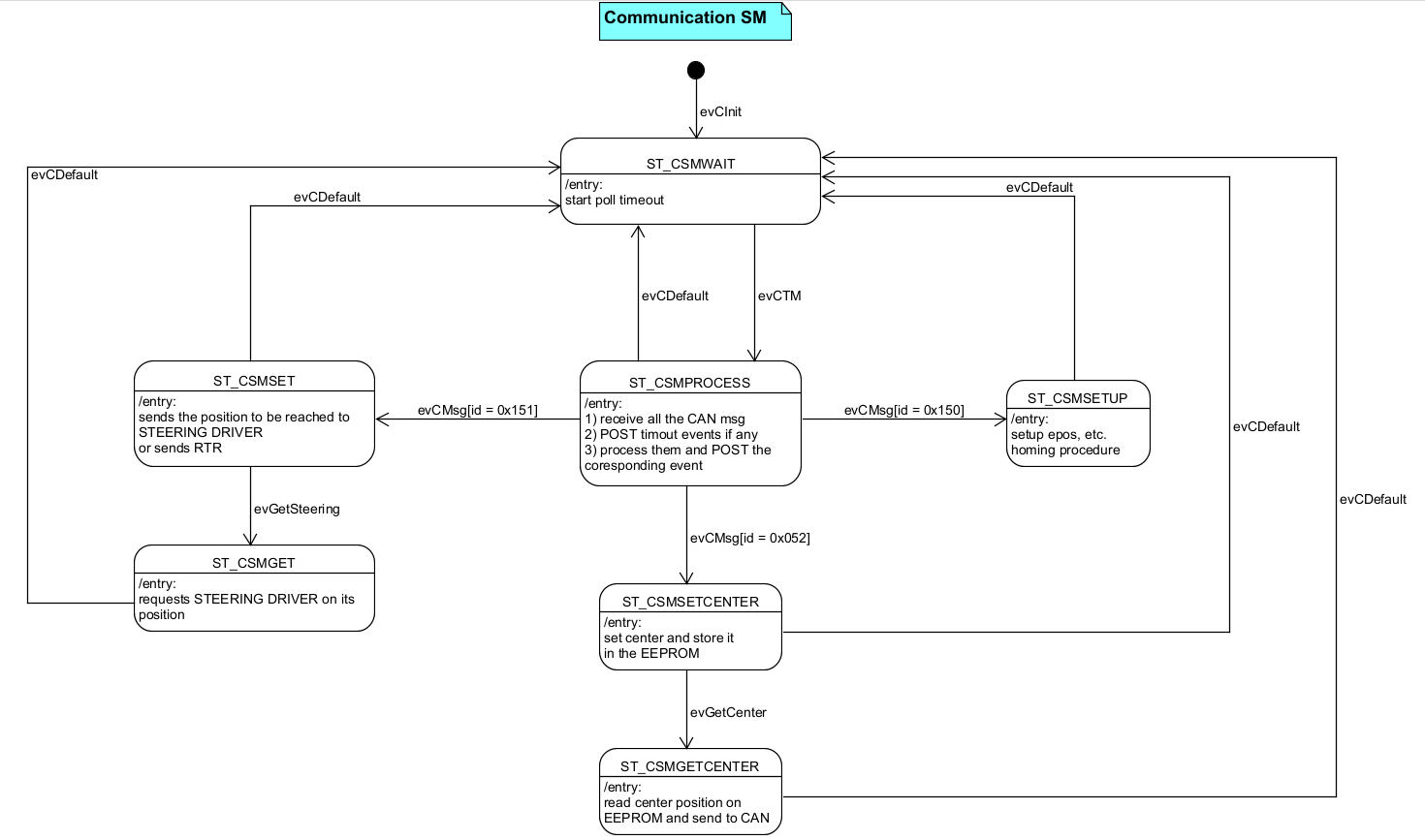


Figure 7 State Machine Communication

Des délais ont du être ajouté avant chaque appelle d’un méthode sepos\_send\_controlworld() ou sepos\_send\_modOffOp(), ces délais sont implémentés dans le POST du xf. M. Sartoretti a confirmé que le epos2 était très lent a traité ses changements de modes.

Dans chacune de ces machines d’état, nous faisons appel aux méthodes de la classe Sepos\_RS232. Cette classe a été codée par Killian Rossier et gère la communication en UART avec le driver du moteur d’entraînement. La construction des différentes méthodes a été faite en suivant les instructions dans la documentation fournie par les professeurs. Le reste a été codé par Maxime Caloz selon la même architecture générale faite par Morgane Porchet pour le *drive control*.

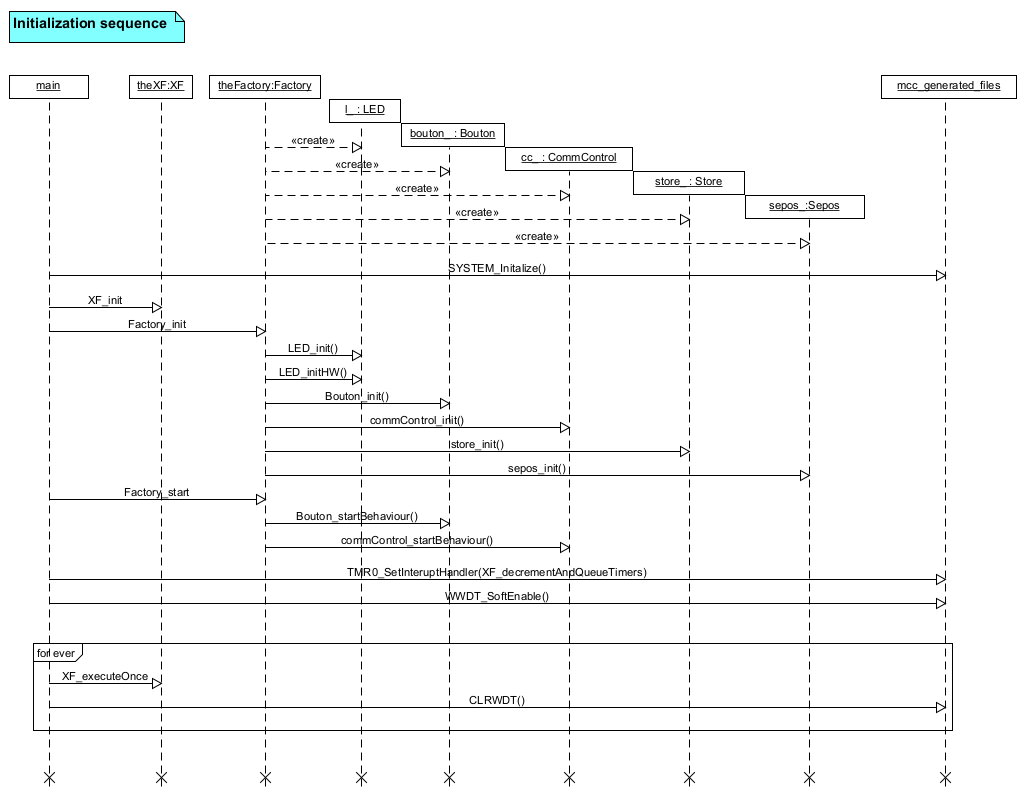
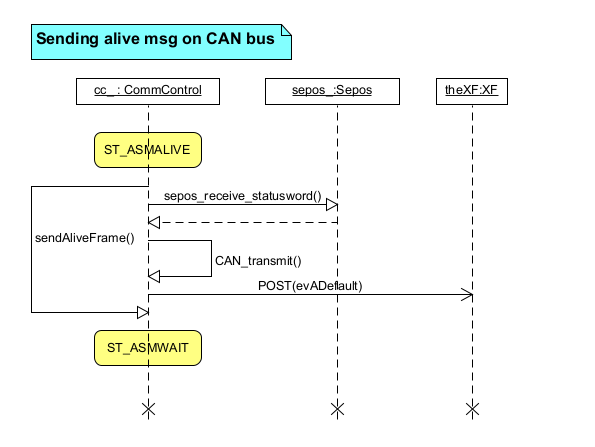
**Ci-dessous le diagramme de séquence du démarrage du code (Figure 8).

Figure : Diagramme de séquence de l'initialisation

L’envoi de la trame alive ce fait comme présenté dans le diagramme suivant (Figure 9).

Figure : Diagramme de séquence de l'envoi de la trame alive.

# Phase de tests

Après avoir programmé le PIC, nous avons procédés à des tests afin de valider le bon fonctionnement de notre code. Cette période de tests s’est faite en plusieurs phases.

Dans un premier temps, nous avons fait clignoter une LED toutes les secondes. De cette manière, nous pouvons nous assurer que le PIC fonctionne bien que ce soit en phase de debug ou non.

Ensuite, nous avons contrôlé que la communication CAN fonctionnait. Pour ce faire, nous avons utilisé le programme *Busmaster* et la base de données fournie par les professeurs (Figure 8). Ces tests nous ont permis de valider le bon fonctionnement de nos trois machines d’état (alive SM, speed SM et commControl SM) présentées dans le chapitre suivant.

Puis, nous avons connecté le driver EPOS4 et le moteur afin d’analyser les messages UART à l’aide du programme *WaveForms* et *EPOS Studio*.

Nous avons, ensuite, connecté notre carte CAN avec celle du Control Module, dans un premier temps. Puis, avec les autres différentes cartes de notre architecture.

Finalement, nous avons fixé et câblé les cartes sur le kart. Dans un premier temps nous avons fais les tests sans charge et ensuite nous avons posé le kart parterre pour ajustement finaux.

La phase de test à été faite avec Adrien Azzalini, Adrien Rey, Aurélien Rithner, Killian Rossier, Maxime Caloz, Morgane Porchet, Sylvestre Van Kappel.

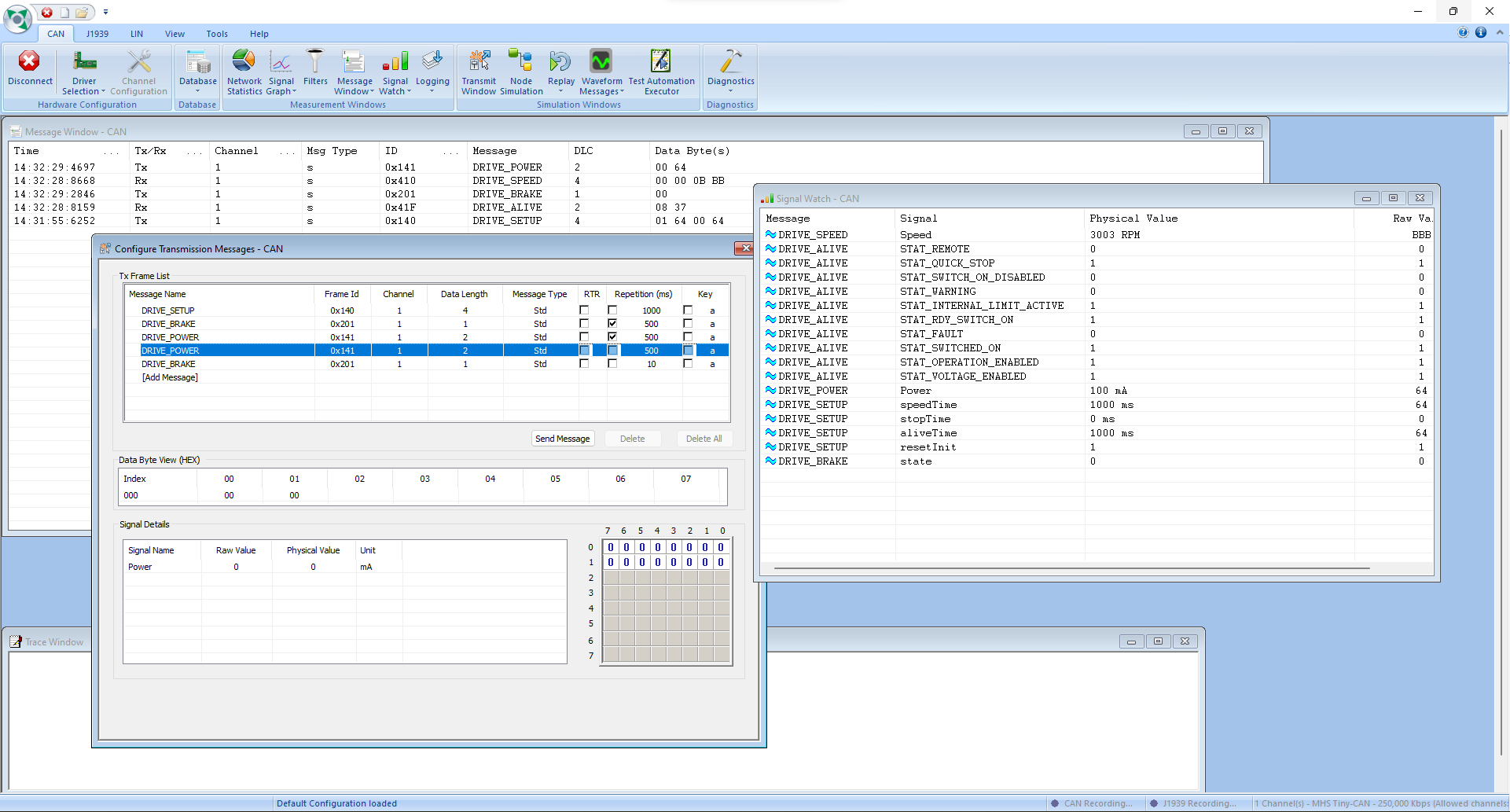


Figure : Programme Busmaster durant la phase de test des messages CAN

# Améliorations et conclusion

Dans la partie communication EPOS, nous avons utilisé des delays. Le code est fonctionnel car ils sont très courts. Cependant, nous pourrions améliorer ce code en utilisant une machine d’état et donc supprimer tous les delays. Nous sommes arrivés à cette réflexion à la suite de problèmes de communication EPOS dans le module steering. Par manque de temps, nous n’avons pas pu implémenter cette modification.

Une autre amélioration serait de stocker les messages reçu sur le CAN dans un buffer et les traiter séparément. Actuellement, nous traitons directement les messages après réception. Le risque ici est de perdre certaines trames. Mais étant donné que nous ne recevons que des messages du contrôleur et que nous lisons les messages chaque 20ms, nous n’avons pas trop de problèmes.

En conclusion, notre architecture de base s’est avérée solide. Nous n’avons pas été confrontés gros problèmes de conception. Nos tests nous ont permis de corriger notre code jusqu’à ce que tout fonctionne correctement.

# Table des illustrations

[Figure 1: Configuration du périphérique TMR0 2](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538594)

[Figure 2: Configuration du périphérique UART1 2](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538595)

[Figure 3: Configuration du System Module 3](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538596)

[Figure 4: Configuration du System Module - WatchDog 3](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538597)

[Figure 5 Diagramme de classe 4](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538598)

[Figure 6 State Machine Alive 5](#_Toc113538599)

[Figure 9: Diagramme de séquence de l'initialisation 7](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538600)

[Figure 10: Diagramme de séquence de l'envoi de la trame alive. 7](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538601)

[Figure 13: Programme Busmaster durant la phase de test des messages CAN 9](https://d.docs.live.net/659b9223512a0f9d/Cours%202eme/Summer%20School/steeringModule/steeringModule_docFinale.docx#_Toc113538602)

# Annexes

Le code a été rendu dans un fichier zip.