

**REPUBLIQUE TUNISIENNE
MINISTRE DE L'ENSEIGNEMENT SUPERIEUR,
DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DE TUNIS EL MANAR**



المعهد العالي للإعلامية

INSTITUT SUPERIEUR D'INFORMATIQUE

RAPPORT DE PROJET DE FIN D'ETUDES

Présenté en vue de l'obtention du
Diplôme : Licence Appliquée en Informatique Industrielle
Option : **Systèmes embarqués**

Par

BARRAK Mohamed Amine

**Implémentation du Protocole CAN Open sur la
plateforme ezlinx**

Entreprise/Organisme d'accueil :



Encadrant à l'entreprise : **M. BARREH Walid**
Encadrant à l'ISI : **Mme BEN HLIMA Ines**

Année Universitaire 2012-2013

A ma chère mère

*Pour tout l'amour et la patience qu'elle m'a offerts.
Qu'elle trouve ici l'expression de ma gratitude et ma fidélité.*

A mon père

*Qui n'a cessé de m'encourager. Tu as toujours été exemplaire dans
ma vie, j'ai beaucoup appris auprès de toi.*

A mon frère et mes sœurs

Avec mes meilleurs vœux de réussite dans leurs études et vies.

A tous mes amis

*En témoignage de ma grande estime et de mon immense
reconnaissance pour tous les conseils et les encouragements qu'ils n'ont
jamais cessé de me prodiguer.*

A tous ceux qui m'ont aidé de près ou de loin

Je dédie ce travail.

Remerciements

Le présent travail a été effectué dans le cadre de mon Projet de Fin d'Etudes. Ce stage a quatre mois chez l'entreprise EBSYS.

Au terme de ce travail, je tiens à exprimer ma gratitude et ma reconnaissance à Monsieur Adel Ghazel, Professeur à l'Ecole Supérieure des Communications de Tunis, qui m'a offert l'opportunité de faire mon stage à EBSYS.

Je remercie, également, mes encadreur Monsieur Walid Barreh, Chef projet à EBSYS, et Madame Ines Ben Hlima, enseignante à l'ISI, pour leurs conseils précieux et leurs remarques constructives.

Mes remerciements les plus distingués sont adressés aux membres du jury qui m'ont fait l'honneur de bien vouloir accepter d'évaluer ce travail.

Un grand merci à toute l'équipe de EBSYS, particulièrement Madame jihen Thabet et Mademoiselle Rim Sari, de m'avoir bien accueilli et de m'avoir incité à mener à bien ce travail.

Finalement, j'exprime mes sincères reconnaissances à l'égard de tous ceux qui ont contribué à ma formation, particulièrement les enseignants de l'ISI.

Sommaire

REMERCIEMENTS	III
SOMMAIRE	IV
LISTE DES FIGURES	VI
LISTE DES TABLEAUX	VIII
LISTE DES ACRONYMES	IX
INTRODUCTION GENERALE	1
CHAPITRE I : PRESENTATION DU CADRE DU STAGE	2
I.1 INTRODUCTION	2
I.2 PRESENTATION DE L'ENTREPRISE D'ACCUEIL.....	2
I.2.1 <i>Domaine de développements</i>	3
I.2.2 <i>Activités de développement</i>	3
I.2.3 <i>Service de consultance</i>	4
I.3 PRESENTATION DU SUJET DE STAGE	4
I.3.1 <i>Contexte du projet</i>	4
I.3.2 <i>Cahier des charges du projet</i>	5
I.3.3 <i>Démarche suivie pour la réalisation du projet</i>	5
I.4 ETUDE DU BUS CAN	6
I.4.1 <i>Définition du BUS CAN</i>	6
I.4.2 <i>Protocole CAN et modèle OSI</i>	7
I.4.3 <i>Aspects physiques du bus CAN</i>	8
I.4.4 <i>Transmission des messages sur le bus CAN</i>	11
I.4.5 <i>Les protocoles basés sur le bus CAN</i>	16
I.5 CONCLUSION	16
CHAPITRE II : ETUDE DU PROTOCOLE CANOPEN.....	17
II.1 INTRODUCTION	17
II.2 PRESENTATION DU PROTOCOLE CANOPEN	17
II.2.1 <i>Concept CANopen et domaines d'application</i>	17
II.2.2 <i>Modèle de communication CANopen</i>	18
II.2.3 <i>Les différents types de communication</i>	18
II.3 STRUCTURE INTERNE D'UN APPAREIL CANOPEN	20
II.3.1 <i>Architecture interne</i>	20

II.3.2	Objets et Dictionnaire Objets.....	21
II.4	INTERFACES DE COMMUNICATION.....	22
II.4.1	Gestion de réseau	23
II.4.2	Protocole de contrôle de l'état des nœuds Heartbeat	27
II.4.3	Messages d'accès au dictionnaire d'objets.....	28
II.5	CONCLUSION.....	36
CHAPITRE III : IMPLEMENTATION DU PROTOCOLE CANOPEN SUR DSP		37
III.1	INTRODUCTION	37
III.2	ORGANIGRAMMES DESCRIPTIFS DU PROTOCOLE CANOPEN.....	37
III.2.1	Organigramme général du CANopen.....	37
III.2.2	Organigramme du protocole NMT.....	39
III.2.3	Organigramme de la fonction Transmettre PDO.....	40
III.2.4	Organigramme de la fonction Recevoir PDO	41
III.2.5	Organigramme de la fonction Client Download SDO.....	42
III.2.6	Organigramme de la fonction Client upload SDO	43
III.2.7	Organigramme de la fonction Serveur Download SDO.....	44
III.2.8	Organigramme de la fonction Serveur upload SDO	45
III.2.9	Organigramme de la fonction Abort SDO Transfert	46
III.3	ENVIRONNEMENTS DE DEVELOPPEMENT HARDWARE ET SOFTWARE	47
III.3.1	Architecture du processeur Blackfin BF548.....	47
III.3.2	Environnement de développement Visual DSP++	51
III.4	RESULTATS D'IMPLEMENTATION.....	52
III.4.1	Utilisation mémoire	52
III.4.2	Test et validation	52
III.5	CONCLUSION.....	57
CONCLUSION GENERALE		58
BIBLIOGRAPHIE		59

Liste des figures

Figure I-1 : Logo de EBSYS	2
Figure I-2: Méthodologie du travail	5
Figure I-3 : Câblage du réseau CAN	6
Figure I-4: Un nœud sur le bus CAN	7
Figure I-5 : relations entre modèle OSI et protocole CAN	7
Figure I-6: Transmission de la trame 011010	9
Figure I-7 : Bit Stuffing.....	9
Figure I-8: Le bit timing.....	10
Figure I-9: Format de la trame de donné.....	11
Figure I-10 : Champ d'arbitrage	12
Figure I-11: Champ de contrôle	12
Figure I-12: Champ de données	13
Figure I-13: Champ de CRC	13
Figure I-14 : Champ d'acquittement	14
Figure I-15 : Format de la trame de requête.....	14
Figure I-16: Format de la trame d'erreur	15
Figure I-17 : Format de la trame de surcharge	15
Figure II-1 : Exemples de domaines d'application du CANopen	18
Figure II-2 : Interaction des couches du protocole CAN	18
Figure II-3 : Modèle Client/Serveur.....	19
Figure II-4 : Modèle Producteur/Consommateur	19
Figure II-5 : Modèle Master/Slave	20
Figure II-6 : Architecture appareil CANopen	20
Figure II-7: Network Management (NMT).....	23
Figure II-8 : Diagramme d'état des esclaves NMT.....	23
Figure II-9 : Message NMT	24
Figure II-10 : Synchronisation Object.....	24
Figure II-11 : Message SYNC.....	25
Figure II-12 : SYNC Time Définition.....	25
Figure II-13 : Time Stamp.....	25

Figure II-14 : Trame Time Stamp	26
Figure II-15 : Emergency Service	26
Figure II-16 : Trame Emergency.....	27
Figure II-17 : Message Heartbeat.....	28
Figure II-18: Communication SDO	29
Figure II-19 : Protocol Download/Upload SDO en mode accéléré	30
Figure II-20 : Description du champ CS en mode accéléré	30
Figure II-21 : Protocol Download SDO segmenté en mode normal	31
Figure II-22 : Description du champ Command Specifier (CS) en mode normal	31
Figure II-23 : Protocol Abort SDO	32
Figure II-25: Communication PDO	33
Figure II-26 : Services PDO.....	33
Figure II-27 : déclenchement pdo	34
Figure II-28 : PDO Mapping.....	34
Figure III-1 : Organigramme général du CANopen.....	38
Figure III-2 : Organigramme NMT	40
Figure III-3 : Organigramme Transmettre PDO	41
Figure III-4 : organigramme Recevoir PDO	42
Figure III-5 : Organigramme Client Download SDO	43
Figure III-6 : Organigramme Client Upload SDO	44
Figure III-7 : Organigramme Serveur Download SDO.....	45
Figure III-8 : Organigramme Serveur Upload SDO	46
Figure III-9 : Organigramme Abort SDO	47
Figure III-10 : schéma synoptique fonctionnel ADSP-BF548.....	47
Figure III-11: Architecture du cœur ADSP-BF548	48
Figure III-12 : Carte mémoire ADSP-BF548.....	49
Figure III-13 : schéma électrique du bus CAN	50
Figure III-14: VisualDSP++.....	51
Figure III-15 : Test de la fonction SYNC	54
Figure III-16 : Test de la fonction Transmettre PDO.....	55
Figure III-17 : Test de la fonction Initiate Download SDO	56
Figure III-18 : Test de la fonction Download SDO Segmented.....	57

Liste des tableaux

Tableau I-1 : Longueur du CAN en fonction du débit et de la durée bit	10
Tableau II-1 : Dictionnaire Objet.....	21
Tableau II-2 : Exemple d'Index channel RS-232	22
Tableau II-3 : Variables du message NMT	24
Tableau II-4: Emergency Error Code.....	27
Tableau II-5 : Variables du message Heartbeat	28
Tableau II-6 : Description des types de transfert SDO	29
Tableau II-7 : Variables du de l'octet CS.....	30
Tableau II-8 : SDO Abort code.....	32
Tableau II-9 : Les objets des SDO	32
Tableau II-10 : Les paramètres du PDO mapping	35
Tableau II-11 : Les paramètres de communication PDO	35
Tableau II-12 : Les objets des PDO	35
Tableau III-1 : Mémoire utilisée par le code CANopen	52
Tableau III-2 : les objets CANopen et leurs COB_ID attribués	53

Liste des acronymes

ALU	Arithmetic and Logic Unit
CAN	Controller Area Network
CiA	CAN in Automation international users and manufacturers group
COB	Communication Object
COB ID	Communication Object Identifier
CRC	Cyclic Redundancy Code
CSMA CD/AMP	Carrier Sense Multiple Acces with Collision Detection and Arbitration Message Priority
DLC	Data Length Code
DMA	Direct Memory Access
DSP	Digital Signal Processor
EMCY	Emergency
LLC	Logical Link Control
MAC	Medium Acces Control
MAC	Multiplier/Accumulator
MDI	Medium Dependent Interface
NMT	Network Management
Node ID	identification du nœud
NRZ	Non Return To Zéro
OSI	Open Systems Interconnection
PDO	Process Data Object
PLS	Physical signalling
PMA	Physical Medium Attach
RPDO	Receive PDO
RSDO	Receive SDO
RTR	Remote Transmission Request
SDO	Service Data Object
SYNC	Synchronisation
TPDO	Transmit PDO
TSDO	Transmit SDO

Introduction générale

Les bus de communication ont été développés à l'origine pour les systèmes embarqués des véhicules automobiles. Actuellement, ils sont utilisés dans plusieurs domaines de l'industrie comme le transport, les équipements mobiles, les équipements médicaux et dans les bâtiments. Leur rôle est d'assurer l'interconnexion de différents équipements électriques du même système.

Les avantages des bus de communications sont nombreux, outre le transport fiable de données, les bus de terrain offre une bonne robustesse contre les perturbations et la réduction des coûts d'installation et de maintenance.

Lors de la conception d'un système, il ne suffit pas de choisir son bus de communication mais il faut surtout choisir le protocole de communication utilisé sur le bus permettant de gérer la transmission fiable des données. Dans notre projet de fin d'études nous nous sommes intéressés en particulier au protocole CANopen. Notre objectif est la mise en œuvre du protocole sur une plateforme DSP du type ezLINX. Pour ce faire, nous avons commencé par étudier les bus CAN et le protocole CANopen pour comprendre le mécanisme d'échange de données et identifier les champs des trames qui trafiquent sur le bus. Ensuite, nous avons procédé à l'implémentation du protocole sur DSP de type Blackfin BF548 en utilisant le software visual DSP++ pour le développement du code. Enfin, nous avons validé notre code à travers des tests d'intégrité.

Notre rapport s'articule en trois chapitres. Dans le premier chapitre, nous allons présenter l'entreprise d'accueil EBSYS, le contexte de notre projet, ainsi que la démarche que nous avons choisie pour la mise en œuvre de notre projet. Le deuxième chapitre décrit la solution du protocole CANopen. Le dernier chapitre s'intéresse à la conception détaillée, nous y présenterons l'environnement de travail logiciel et matériel, ainsi que le test du code en validant les résultats obtenus.

Chapitre I

Présentation du cadre du stage

I.1 Introduction

Le but de ce chapitre est de présenter le cadre général de notre projet de fin d'études. La première partie de ce chapitre est consacrée à la présentation de la société d'accueil et ses domaines d'activité. Le cahier des charges ainsi que la démarche de conduite du projet seront détaillés dans la deuxième partie. Dans la dernière partie du chapitre, nous présenterons une étude détaillée sur le bus CAN qui constitue le contexte général de notre projet en citant ses différentes caractéristiques ainsi que ses couches applicatives.

I.2 Présentation de l'entreprise d'accueil

EBSYS est la dénomination commerciale de la société «Embedded Systems Technology» [1]. C'est une société de droit privé Tunisien totalement exportatrice et a pour mission le développement de solutions logicielles et matérielles pour des systèmes électroniques embarqués principalement pour des applications de Télécommunication EBSYS a démarré ses activités en Tunisie en septembre 2001 et dispose d'un centre de développement sis au Pôle El Ghazala des Technologies de Communications – Ariana Nord – Tunis. EBSYS est un partenaire technologique de la firme Américaine Analog Devices, leader mondial dans la technologie des circuits électroniques (circuits de traitement des signaux analogiques et numériques pour des applications basses et hautes fréquences). EBSYS fait partie des centres de développement de programmes rattachés à la Division « Software and Systems Technology » d'Analog Devices. Les activités d'EBSYS jouent un double rôle : d'abord la promotion de nouvelles technologies tunisiennes et puis la contribution au développement d'autres technologies de pointe en collaboration avec les autres centres de programmes d'Analog Devices dans le monde. Le sigle d'EBSYS est donné par la figure I-1.



Figure I-1 : Logo de EBSYS

I.2.1 Domaine de développements

Les domaines de développement d'EBSYS couvrent des activités de mise en œuvre de solutions logicielles sur DSP et la conception de systèmes matériels pour les applications suivantes.

I.2.1.1 Systèmes de communication bande étroite et large bande

Ils sont des systèmes avec une spécialisation dans les domaines suivants :

- Systèmes et réseaux de communication sur ligne d'énergie électrique : réseaux d'accès et réseaux à l'intérieur des bâtiments.
- Systèmes et réseaux de communication sans fils en boucle locale (PANs et LANs).
- Modems téléphoniques.

I.2.1.2 Multimédia, téléphonie, réseaux et gestion des processus industriels

Ce sont avec une spécialisation dans les domaines suivants :

- Traitement audio et vidéo.
- Services de gestion de réseau (NMS).
- Services à valeur ajoutée sur RTCP : sécurisation de l'accès téléphonique, répondeur identification de l'appelant.
- Gestion de flotte par GPS.
- Télé relève des compteurs électroniques.

I.2.2 Activités de développement

L'activité de développement consistent sur :

- Détermination des spécifications de conception de produits.
- Ingénierie des systèmes de communications.
- Développement logiciel.
- Prototypage de nouveaux produits et positionnement sur le marché.
- Support technique des clients.

I.2.3 Service de consultance

Les services de consultance de notre société sont :

- Caractérisation des signaux d'entrée / sortie.
- Partitionnement du traitement analogique / numérique.
- Contraintes de conception : temps de traitement, format de données.
- Définition des algorithmes de traitement du signal et analyse de la complexité.
- Conception assistée par ordinateur des cartes électroniques et analyse des performances.
- Implantation logicielle et partitionnement de codes (adaptation aux technologies DSP).
- Intégration de codes et intégration système.
- Tests expérimentaux et analyse des performances physiques et logiques.

I.3 Présentation du sujet de stage

Dans cette partie, nous allons présenter le contexte de notre projet, le cahier des charges et la démarche que nous allons adopter pour la bonne conduite du projet.

I.3.1 Contexte du projet

De nos jours, les réseaux de terrain sont intégrés dans tous les domaines de l'industrie. Ils permettent l'interconnexion entre plusieurs entités d'un même système, exemples : appareils de mesure, capteurs, microcontrôleurs, actionneurs, mémoires, etc.

Dans les réseaux de terrain, les débits de transmission de données et les distances de communication sont assez faibles comparativement aux autres types de réseaux, néanmoins, la contrainte temps réel est prioritaire. Le support de transmission est généralement un câble coaxial blindé ou une paire torsadée travaillant en mode différentiel ou encore une fibre optique pour plus d'immunité contre le bruit.

Une topologie en bus (un modèle commun de transmission) est généralement adoptée pour faciliter la mise en place, l'évolution et l'extension des systèmes. Les domaines d'application des bus de terrain s'étendent à des améliorations continues de tous ces systèmes.

Outre le transport fiable de données, les bus de terrain offre une bonne robustesse contre les perturbations, le contrôle de la décentralisation du système et la réduction des coûts d'installation et de maintenance.

Pour la mise en œuvre d'un bus de terrain, un protocole de communication doit être développé afin de gérer la transmission fiable des données. Dans notre projet de fin d'études nous nous sommes intéressés en particulier au protocole CANopen.

I.3.2 Cahier des charges du projet

EBSYS dispose déjà des couches qui gèrent la communication (envoi et réception des trames) sur le bus CAN. Elle envisage implémenter le CANopen qui est le protocole de la couche supérieure de la base CAN (Controller Area Network) sur la plate-forme ezLINX qui dispose d'un bus CAN, afin de gérer le codage des données sur les trames d'échange des différents appareils connectés à un bus CAN.

Dans ce cadre, après avoir vérifié la présence des moyens matériels et logiciels nécessaires, il nous a été demandé, ainsi, de faire une démonstration de la possibilité de mise en œuvre du protocole CANopen.

I.3.3 Démarche suivie pour la réalisation du projet

Pour la réalisation de ce projet, nous avons opté pour une méthodologie de travail donnée par la figure I-2.

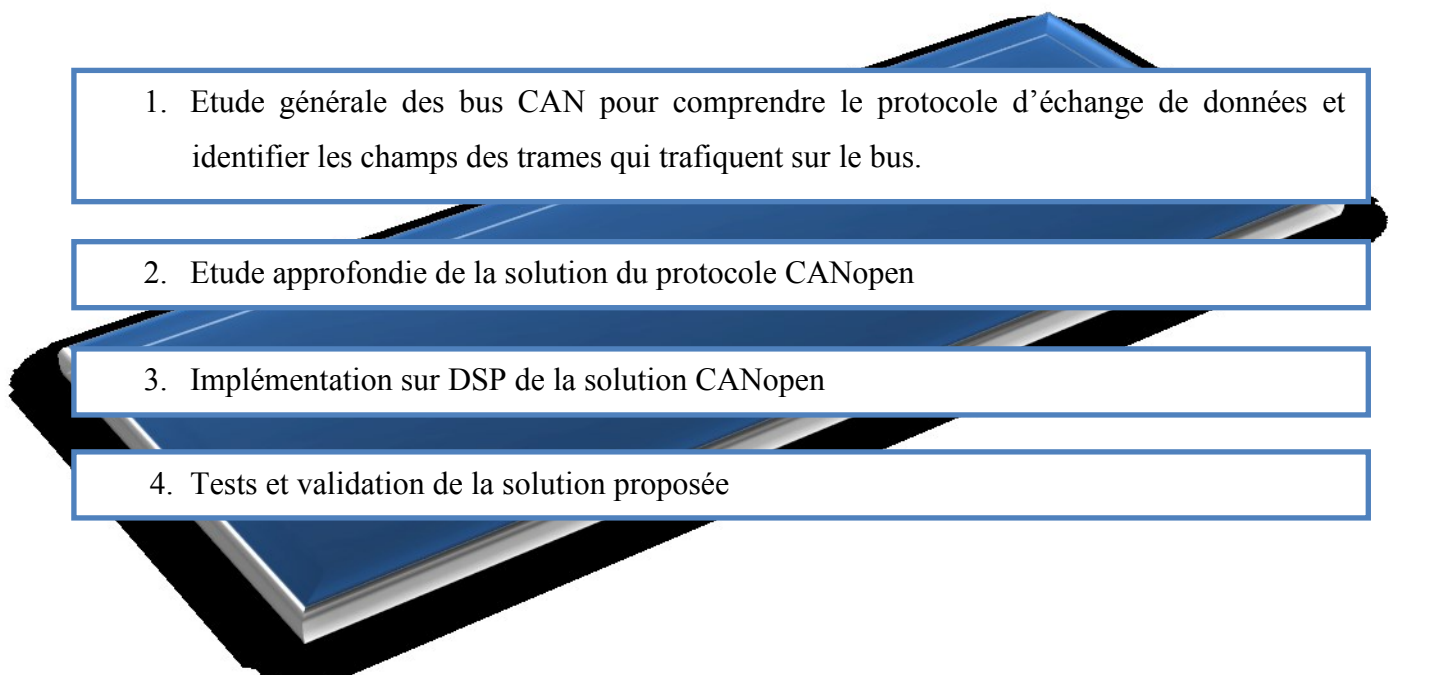
- 
1. Etude générale des bus CAN pour comprendre le protocole d'échange de données et identifier les champs des trames qui trafiquent sur le bus.
 2. Etude approfondie de la solution du protocole CANopen
 3. Implémentation sur DSP de la solution CANopen
 4. Tests et validation de la solution proposée

Figure I-2: Méthodologie du travail

I.4 Etude du BUS CAN

I.4.1 Définition du BUS CAN

Il existe plusieurs variétés de bus de terrain, les plus courants sont : le bus CAN, le bus LIN et le bus ASi.

Le bus CAN (Control Area Network) est un bus de communication série de type multi-maîtres, asynchrone qui supporte des systèmes embarqués temps réel avec un haut niveau de fiabilité [2]. Il est assuré par un mécanisme d'erreur performant, tels qu'on peut en trouver dans les automobiles, et ceci avec un très haut niveau d'intégrité au niveau des données.

Le bus CAN a été standardisé par l'ISO dans les normes 11898 pour les applications à hauts débits (125Kbps-1Mbps) et ISO 11519 pour les applications à bas débits (<125Kbps).

La transmission des données sur bus CAN est effectuée sur une paire filaire différentielle comme c'est indiqué par la figure I-3 [3]. La ligne est donc constituée de deux fils : CAN L (CAN LOW) et CAN H (CAN HIGH). Le montage différentiel permet de réduire les interférences et donc assure une immunité électromagnétique car les deux lignes du bus sont toutes les deux affectées de la même manière par un signal perturbateur. La ligne du bus doit se terminer par des résistances de 120 Ω (minimum 108 Ω , maximum 132 Ω) à chacun des bouts.

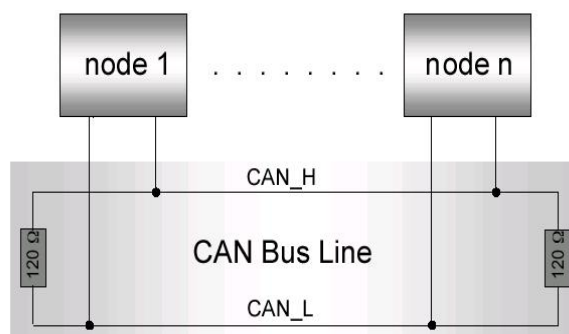


Figure I-3 : Câblage du réseau CAN

Un nœud du bus CAN nécessite pour son fonctionnement au sein du réseau un microcontrôleur et un contrôleur CAN. Il est connecté à l'instrument de transmission qui transforme les bits en tensions (le Transceiver), en utilisant la ligne de transmission Tx et la ligne de réception Rx. Les tensions de référence d'alimentation du Transceiver sont de 5 volts. Ceci est détaillé dans la figure I-4.

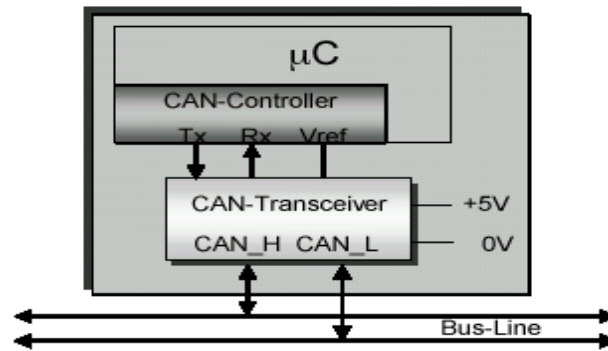


Figure I-4: Un nœud sur le bus CAN

Le bus CAN possède plusieurs propriétés, il permet notamment la hiérarchisation des messages, la souplesse de configuration, le fonctionnement dans un contexte multi-maître et la réception de multiples sources avec synchronisation temporelle, la garantie des temps de latence, la retransmission automatique des messages altérés dès que le bus sera en repos, ainsi que la déconnexion automatique des nœuds défectueux.

I.4.2 Protocole CAN et modèle OSI

La figure I-5 décrit l'interaction du protocole CAN avec les sept couches du modèle OSI (Open System Interconnexion) [4]. Le protocole CAN ne couvre que la totalité de la couche 2 (Data Link layer : couche de communication de données), une partie importante de la couche 1 (Physical layer : couche physique) et la couche application (Logiciel) dans laquelle est implémenté le protocole qui gère la communication des données à travers le bus.

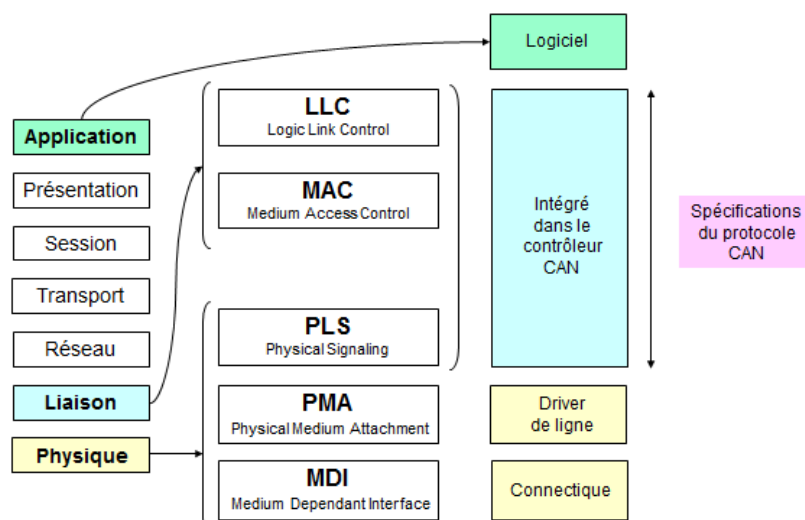


Figure I-5 : relations entre modèle OSI et protocole CAN

Nous détaillons dans ce qui suit les fonctionnalités des couches du modèle OSI qui sont supportées par le protocole CAN.

1.4.2.1 La couche physique

Cette couche définit comment le signal est transmis sur le support physique. Cette couche gère la représentation du bit (codage, timing, synchronisation), et définit les niveaux électriques, optiques,... des signaux ainsi que le support de transmission. Elle comprend trois sous-couches :

- PLS (Physical signalling) : responsable du codage /décodage des bits, de la durée du bit et de la synchronisation.
- PMA (Physical Medium Attach) : elle a les caractéristiques de l'émetteur/récepteur physique.
- MDI (Medium Dependent Interface) : Il s'agit du connecteur entre la couche physique et la couche supérieure.

Le protocole CAN ne décrit que la représentation détaillée du bit (Physical Signalling), mais pas le moyen de transport et les niveaux des signaux de telle sorte qu'ils puissent être optimisés selon l'application.

1.4.2.2 La couche liaison de données

Elle est subdivisée en deux sous-couches :

- MAC (Medium Acces Control) : elle est responsable de la mise en trame du message, l'arbitrage, l'acquittement, la détection et la signalisation des erreurs.
- LLC (Logical Link Control) : elle est responsable du filtrage des messages, la notification des surcharges et la procédure de recouvrement des erreurs.

La sous couche MAC présente le cœur du protocole CAN. Elle a pour fonction de présenter les messages reçus à la sous-couche LLC (Logical Link Control) et d'accepter les messages avant d'être transmis vers la sous couche LLC.

1.4.3 Aspects physiques du bus CAN

Cette partie concerne les paramètres physiques de la liaison entre les nœuds connectés sur un bus CAN ainsi que la longueur du bus CAN [5].

1.4.3.1 Le codage NRZ

Les bits qui transitent sur le bus sont codés avec la technique NRZ (Non Return To Zéro). Pendant la durée totale du bit, le bit 1 est représenté par un état significatif (par

exemple, une tension clairement positive), et le bit 0 par un autre état significatif (par exemple, une tension clairement négative). Il n'existe pas d'état intermédiaire.

La figure I-6 représente un exemple de transmission d'une suite binaire sur le bus CAN.



Figure I-6: Transmission de la trame 011010

I.4.3.2 Le Bit stuffing

Une des caractéristiques du codage NRZ est que le niveau du bit est maintenu pendant toute sa durée. Cela pose des problèmes de fiabilité si un grand nombre de bits identiques se succèdent. La technique du Bit Stuffing impose au transmetteur d'ajouter automatiquement un bit de valeur opposée lorsqu'il détecte 5 bits consécutifs dans les valeurs à transmettre.

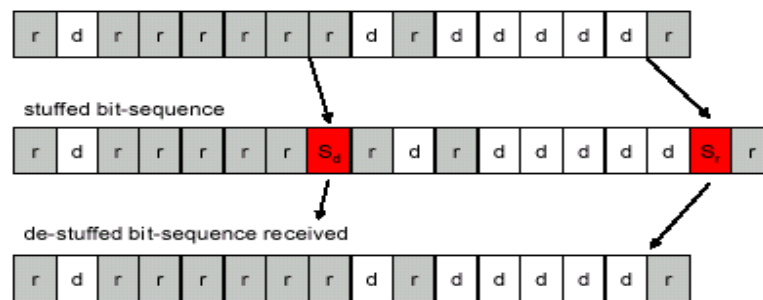


Figure I-7 : Bit Stuffing

Après l'ajout des bits stuffing, le message comporte plus de bits à transmettre ce qui permet une meilleure synchronisation des nœuds.

I.4.3.3 Le Bit timing

Le bit timing permet de garantir les contraintes temps réels sur la base des notions suivantes [6]:

- Nominal Bit Rate : c'est le nombre de bits par seconde transmis par un émetteur idéal en l'absence de resynchronisation.
- Nominal Bit Time : c'est l'inverse du Nominal Bit Rate. Il est divisé en quatre segments qui ne se recouvrent pas (figure I-8):
 - SYNC_SEG (Synchronization Segment) : cette partie est utilisée pour synchroniser les divers nœuds du bus.

- PROG_SEG (Propagation Time Segment) : cette partie sert à compenser les retards temporels physiques dans le réseau.
- PHASE_SEG1, PHASE_SEG2 (Phase Buffer Segment 1 et Phase Buffer Segment 2) : ces segments sont utilisés pour compenser les erreurs de synchronisation.

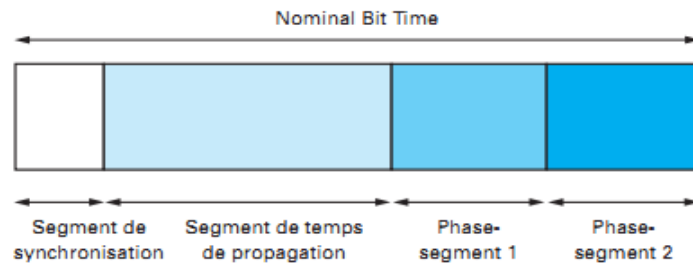


Figure I-8: Le bit timing

I.4.3.4 La longueur du bus

La longueur du bus dépend du délai de propagation sur les lignes physiques du bus [7], de la différence du quantum de temps, de la différence de cadencement des oscillations des nœuds et de l'amplitude du signal qui varie en fonction de la résistance du câble et de l'impédance d'entrée des nœuds. Le tableau I-1 donne la variation de la longueur du CAN en fonction du débit et de la durée bit.

Débit	Longueur	Durée d'un bit
1 Mbit/s	30 m	1 μ s
800 Kbit/s	50 m	1,25 μ s
500 Kbit/s	100 m	2 μ s
250 Kbit/s	250 m	4 μ s
125 Kbit/s	500 m	8 μ s
62,5 Kbit/s	1000 m	16 μ s
20 Kbit/s	2500 m	50 μ s
10 Kbit/s	5000 m	100 μ s

Tableau I-1 : Longueur du CAN en fonction du débit et de la durée bit

Il est important de noter que n'importe quel module (Nœud) connecté sur un bus CAN doit pouvoir supporter un débit d'au moins 20 kbit/s.

Pour une longueur de bus supérieure à 200 mètres il est nécessaire d'utiliser un optocoupleur et pour une longueur de bus supérieure à 1 kilomètre il est nécessaire d'utiliser des systèmes d'interconnexion tels que des répéteurs ou des ponts.

I.4.4 Transmission des messages sur le bus CAN

La communication sur le bus CAN se fait essentiellement par la diffusion de l'information [8]. Chaque station connectée sur le réseau écoute le trafic des trames transmises par les stations émettrices. Elle a le choix d'agir au message arrivé ou non selon sa programmation. Le protocole CAN autorise certaines stations d'accéder simultanément au bus, il possède un système rapide et fiable d'arbitrage qui détermine la station émettrice en premier. Ce mécanisme est géré par la méthode CSMA CD/AMP (Carrier Sense Multiple Acces with Collision Detection and Arbitration Message Priority).

Sur le bus CAN il y a quatre différentes trames :

- **Trame de données (DATA FRAME)** : elle permet de transporter les données d'un émetteur vers un ou plusieurs récepteurs.
- **Trame de requête (REMOTE FRAME)** : elle est générée par un nœud pour demander la transmission d'une trame de données avec le même identificateur.
- **Trame d'erreur (ERROR FRAME)** : elle est transmise par une unité lorsqu'elle détecte une erreur sur le bus.
- **Trame de surcharge (OVERLOAD FRAME)** : elle est utilisée pour générer un retard supplémentaire entre les trames de données ou les trames de requête.

I.4.4.1 Trame de données

La trame de données est constituée de différents champs. Son format est donné par la figure I-9.



Figure I-9: Format de la trame de données

I.4.4.1.1 Début de trame

Ce champ indique le début d'une trame de données. Il s'agit d'un seul bit dominant. La transition du bit SOF permet la synchronisation de tous les nœuds connectés sur le bus lors de la transmission, afin de donner l'accès au nœud qui a commencé la transmission en premier.

I.4.4.1.2 Champ d'arbitrage

Le champ d'arbitrage est composé de 11 bits d'identification (ID) pour la version standard CAN 2.0A et 29 bits pour la version étendue CAN 2.0B suivis par le bit RTR (Remote Transmission Request) qui est dominant. Ce champ sert d'identifiant pour la donnée transportée dans le champ de données.

La figure I-10 explique un exemple d'arbitrage de trois stations synchronisées ensemble sur le bit SOF et comment accèdent-elles au bus CAN. Le mécanisme d'arbitrage permet de comparer les bits du champ d'arbitrage du bit du poids le plus fort jusqu'au bit du poids faible, en connaissant que le bit dominant est 0 et le bit récessif est 1.

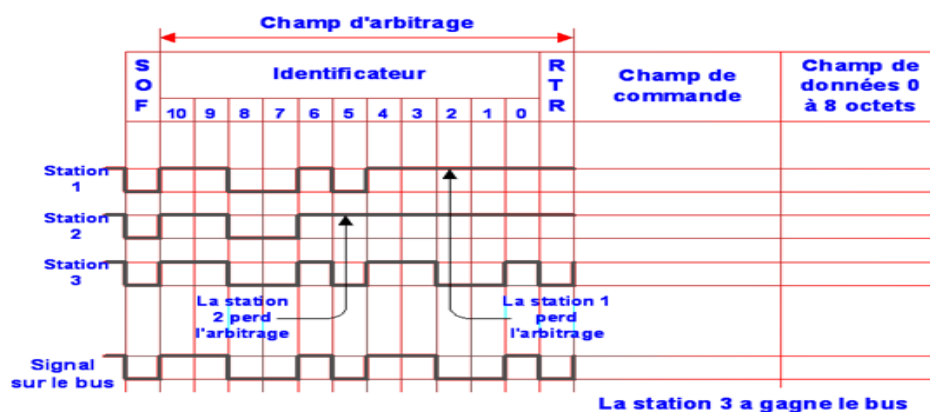


Figure I-10 : Champ d'arbitrage

I.4.4.1.3 Champ de contrôle

Le champ de contrôle est constitué de 6 bits (voir figure I-11), ils sont répartie comme suit : les 2 premiers sont des bits réservés et les 4 bits suivants constituent le Data Length Code (DLC) qui indique le nombre d'octets dans le champ de données. Il y a 9 valeurs possibles pour le DLC car la taille maximale du champ de données est 8 octets (1000 en binaire).

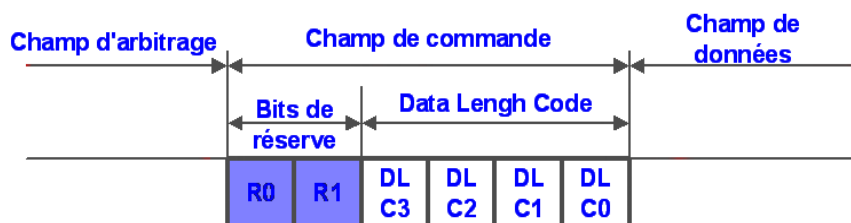


Figure I-11: Champ de contrôle

I.4.4.1.4 Champ de données

Ce sont les données envoyées par une trame de données. Il peut contenir des données de 0 jusqu'à 8 octets (voir figure I-12). Chaque octet est transmis avec le bit du plus fort poids en premier.

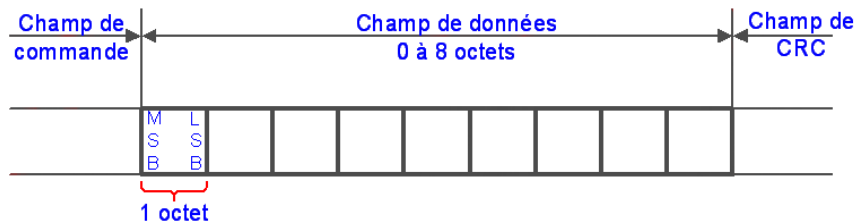


Figure I-12: Champ de données

I.4.4.1.5 Champ de CRC

Ce champ est composé de 15 bits qui indiquent la séquence du CRC et un bit récessif appelé CRC Delimiter qui indique la fin du champ, comme c'est indiqué par la figure I-13.

La séquence de CRC (Cyclic Redundancy Code) permet de vérifier l'intégrité des données transmises. Les bits utilisés dans le calcul du CRC sont ceux du SOF, du champ d'Arbitrage, du champ de Control et du champ de données.

Le CRC est un polynôme calculé de la même manière par l'émetteur et par le récepteur de la trame : le message est vu par l'algorithme du bus CAN comme un polynôme qui est divisé par le polynôme $X^{15}+X^{14}+X^{10}+X^8+X^7+X^4+X^3+1$ et le reste de cette division est la séquence CRC transmise avec le message.

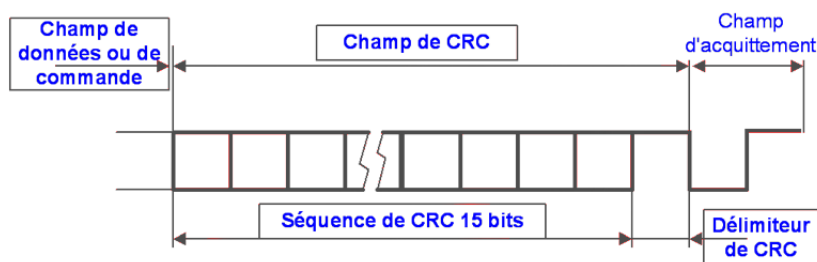


Figure I-13: Champ de CRC

I.4.4.1.6 Champ d'acquittement

Le champ ACK est composé de 2 bits, un bit appelé ACK Slot et un autre récessif appelé ACK Delimiter (voir figure I-14). Le bit ACK Slot est récessif si un nœud en train de transmettre et il est dominant si un nœud reçoit correctement un message.

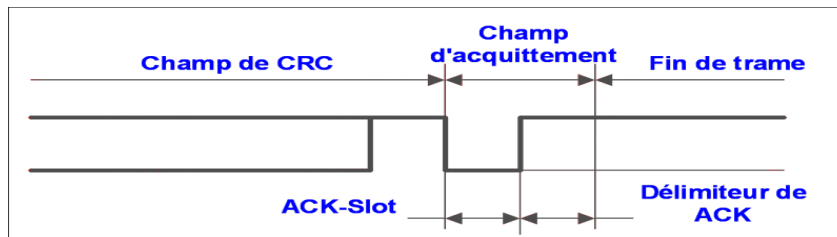


Figure I-14 : Champ d'acquittement

I.4.4.1.7 Fin de trame

Chaque trame de données est terminée par un drapeau formé par une séquence de 7 bits récessifs qui indiquent la fin de la trame.

I.4.4.2 Trame de requête

La trame de requête est créée pour qu'une station puisse demander de l'information d'une autre. Elle comporte six champs, à savoir : début de trame, champ d'arbitrage, champ de contrôle, champ de CRC, champ d'acquittement et la fin de trame. La figure I-15 montre les différents champs de la trame de requête.

Il existe deux différences entre une trame de requête et une trame de données :

- Le bit RTR est transmis comme un dominant dans la trame de données et comme un récessif dans la trame de requête.
- La trame de requête ne contient pas de données dans le champ de données.

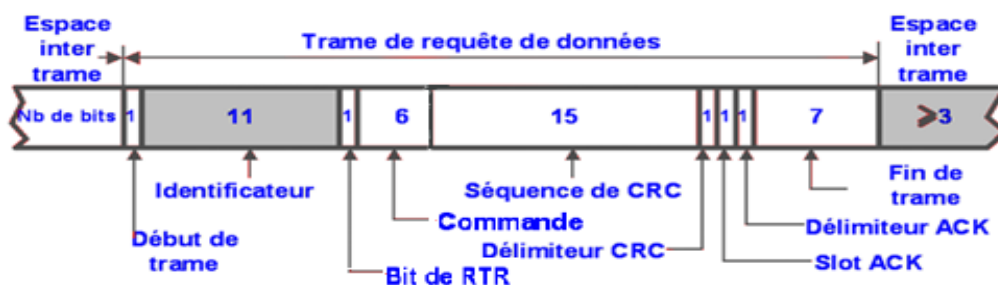


Figure I-15 : Format de la trame de requête

I.4.4.3 Trame d'erreur

Le protocole CAN implémente cinq mécanismes d'erreurs : le bit d'erreur, le Stuff d'erreur, le CRC d'erreur, le Format d'erreur et l'ACK erreur. Le protocole CAN dispose d'un système de gestion des erreurs locales.

Le Stuff erreur est généré lorsqu'il y a 6 bits consécutifs de mêmes valeurs qui devraient être codés avec la méthode du 'bit Stuffing'. Le CRC d'erreur est généré lorsque le récepteur

d'une trame calcule le CRC et le trouve différent de l'émetteur. Le Format d'erreur est généré lorsqu'un bit connu par une valeur donnée est reçu à une valeur différente. L'ACK erreur est généré lorsque le transmetteur détecte une erreur d'acquiescement ou le bit ACK Slot n'est pas dominant.

Comme le montre la figure I-16, une trame d'erreur est constituée de deux parties : drapeau d'erreur (Error flag) émis par les nœuds du bus et un délimiteur d'erreur. Le drapeau d'erreur est constitué de six bits dominants consécutifs (dans ce cas on viole la règle du bit stuffing) qui écrasent les données contenues dans la trame de données. La trame d'erreur provoque la retransmission de la trame qui a rencontré une erreur.

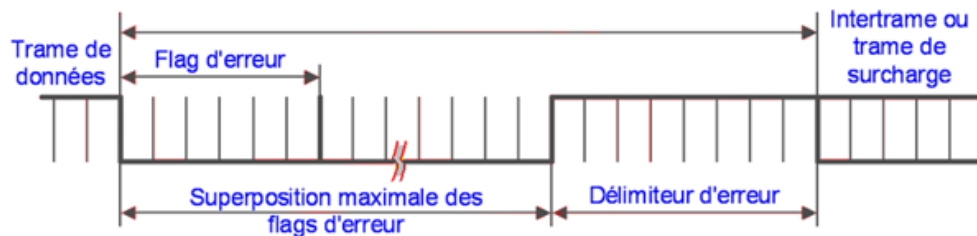


Figure I-16: Format de la trame d'erreur

Remarque : Des travaux de recherche ont montré que le taux d'erreurs binaire du protocole CAN est très faible : 1 erreur non détectée pour 1000 années de fonctionnement.

1.4.4.4 Trame de surcharge

La trame de surcharge indique qu'une station est surchargée pendant un certain temps. Afin de ne pas bloquer le bus indéfiniment, seules deux trames de surcharge consécutives peuvent être générées pour retarder les trames de données ou les trames de requête.

Comme le montre la figure I-17, la trame de surcharge se compose de deux champs : champ de drapeau de surcharge et champ délimiteur.

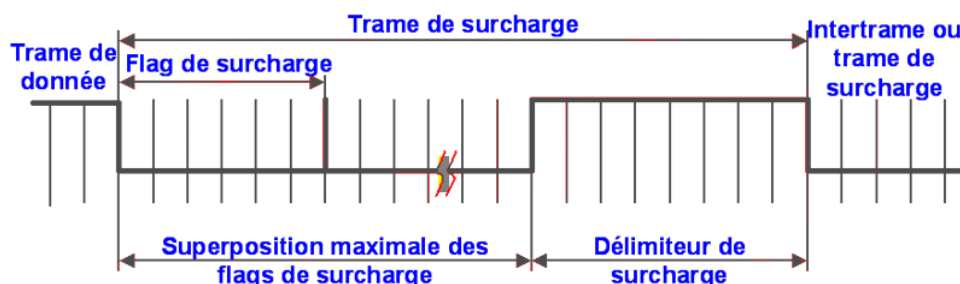


Figure I-17 : Format de la trame de surcharge

I.4.5 Les protocoles basés sur le bus CAN

Dès la création du groupe CiA (CAN in Automation), les membres ont décidé de créer un système ouvert sur la base du protocole CAN [9]. Ce fut la naissance des couches applicatives CAN (CAN Application Layers ou CAL), puis un peu plus tard le CANopen. Ces couches applicatives sont implémentées en langage C.

Le but de l'architecture de ces couches est de cacher aux personnes responsables de la programmation des applications du réseau les détails de la gestion de base de la communication CAN et de leur fournir un premier pas d'implémentation de système à variables partagées.

A ce jour, l'usage des couches applicatives est plus implémenté aux milieux industriels. Pour le CAN, il en existe quatre principales qui sont :

- CAL/CANopen du CiA,
- DeviceNet de la société Allen Bradley,
- SDS (Smart Distributed System) de la société Honeywell,
- OSEK Open systems and interfaces for distributed electronics in cars

Il y en a d'autres qui sont plus dédiées à des applications spécifiques : les CANKingdom (de Kvaser), J1939 (de SAE-Society of automotive engineers aux USA), MMS...

I.5 Conclusion

Dans ce chapitre, nous avons essayé de mettre en évidence le cadre de réalisation du projet en présentant l'organisme d'accueil EBSYS, le contexte du projet de stage et la méthodologie du travail adopté pour réaliser ce projet. Il s'agit du contexte des bus de terrain utilisés dans les systèmes de communication numérique permettant l'échange des données entre plusieurs dispositifs, ou entre un dispositif et ses périphériques. Dans la dernière partie du chapitre nous avons présenté une étude approfondies du bus CAN. A la fin de ce chapitre nous avons cité les protocoles basés sur le bus CAN. Le CANopen en fait partie, il s'agit du protocole que nous avons implémenté dans le cadre de notre projet de fin d'études. L'étude approfondie de ce protocole fera l'objet du prochain chapitre.

Chapitre II

Etude du protocole CANopen

II.1 Introduction

Le deuxième chapitre est dédié à l'étude approfondie du protocole CANopen afin de dériver les différents mécanismes de transfert de données.

Dans la première partie du chapitre nous allons présenter l'historique du CANopen et ses domaines d'application ainsi que le modèle de communication CANopen. Le détail de l'architecture interne d'un appareil CANopen ainsi que les concepts d'Objets et du Dictionnaire Objet feront le but de la deuxième partie du chapitre. Dans la dernière partie, nous allons détailler les interfaces de communication avec le Bus CAN, nous allons parler en particulier des messages de gestion de réseau, de contrôle de l'état des nœuds et de l'accès au Dictionnaire Objets.

II.2 Présentation du protocole CANopen

Dans cette section nous allons présenter en premier lieu le concept général du CANopen et ses domaines d'application, ensuite, nous allons décrire le modèle de communication CANopen et les différents types de communication entre deux dispositifs CAN supportant le protocole CANopen.

II.2.1 Concept CANopen et domaines d'application

CANopen définit un protocole de communication pour les systèmes d'automatisation industrielle distribués basés sur le bus CAN qui permet de raccorder jusqu'à **127 appareils** sur un même réseau. Ce qui offre la possibilité d'avoir l'accès au bus à n'importe quel moment. Il définit exactement comment échanger des données entre les équipements connectés et comment chaque dispositif doit interpréter ces données.

CANopen a été développé au sein du groupe CiA (CAN in Automation international users and manufacturers group) [10], il est normalisé depuis Décembre 2002 sous CENELEC EN 50325-4. Peu de temps après sa sortie initiale en 1996, CANopen a trouvé une large application, notamment en Europe, où il est considéré comme la principale norme pour les

systèmes industriels et embarqués basés sur CAN. CANopen est actuellement utilisé dans divers domaines : automobile, agricole, industriel (ascenseurs, escaliers roulants, motion control) et médical (rayons X, salles d'opérations), etc.



Figure II-1 : Exemples de domaines d'application du CANopen

II.2.2 Modèle de communication CANopen

Le modèle de communication CANopen [11] peut être décrit de la même façon que le modèle de référence OSI décrit dans le premier chapitre. Le CANopen représente une couche d'application standardisée et un profil de communication. Il interagit avec les autres couches du modèle de communication comme c'est indiqué dans la figure II-2. Les spécifications des couches physiques et liaison de données CAN sont décrites dans la section I.4.2 du premier chapitre. La figure II-2 décrit la communication sur les différentes couches du protocole CAN.

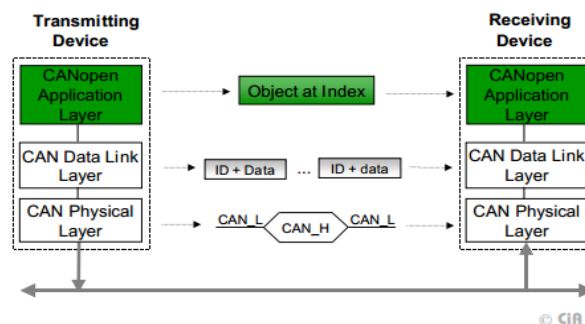


Figure II-2 : Interaction des couches du protocole CAN

II.2.3 Les différents types de communication

Sur la couche application CANopen, les nœuds échangent des objets de communication selon un type de communication bien déterminé. Il existe trois modèles de communication : Client/Serveur, Producteur/Consommateur et Maître/Esclave.

II.2.3.1 Modèle Client/Serveur

Dans ce modèle, il s'agit d'une relation d'échange entre un client et un serveur unique. Le client envoie une requête (Upload/Download) qui déclenche le serveur pour effectuer une certaine tâche. Après avoir terminé la tâche, le serveur répond à la demande de sorte que le client obtient une confirmation (voir figure II-3).

Le modèle Client/Serveur est généralement utilisé pour transférer des données supérieures à 8 octets. En fait, les données à transmettre sont envoyées segment par segment. Le client qui reçoit les données peut envoyer un ou plusieurs confirmations.

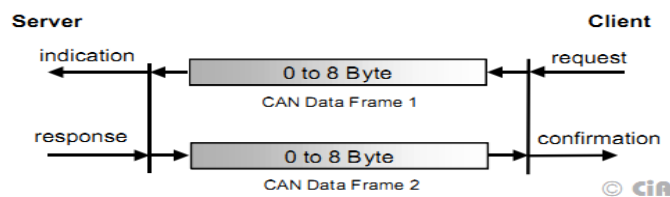


Figure II-3 : Modèle Client/Serveur

II.2.3.2 Modèle Producteur/Consommateur

Le modèle Producteur/Consommateur donné par la figure II-4, décrit les possibilités de diffusion de données sur un réseau CAN. En effet, chaque élément du réseau est en écoute, après avoir reçu le message, l'élément décide s'il l'accepte ou non.

Ce modèle permet de transmettre des messages (Push model) et de demander des messages (Pull model).

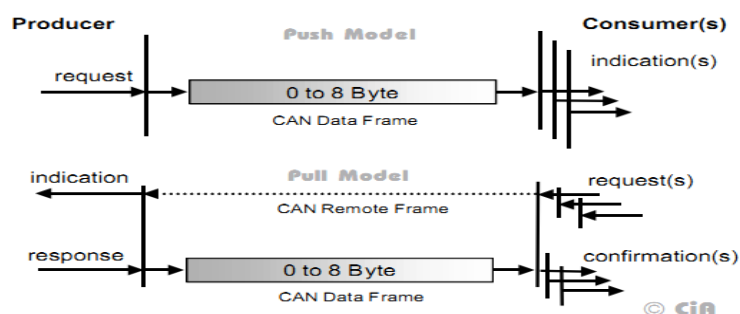


Figure II-4 : Modèle Producteur/Consommateur

II.2.3.3 Modèle Maître/Esclave

Dans le modèle de communication Maître/Esclave, seul le maître peut commencer une communication et l'esclave est toujours en attente d'une requête de communication du maître, comme c'est décrit par la figure II-5.

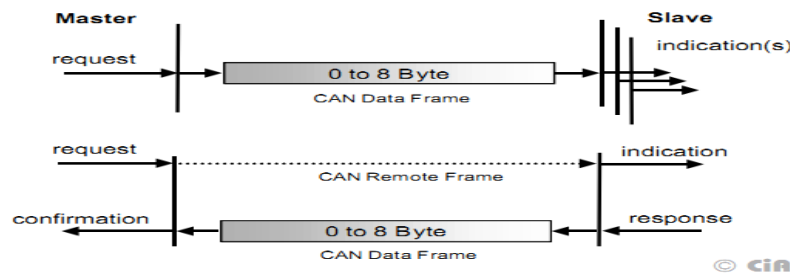


Figure II-5 : Modèle Master/Slave

II.3 Structure interne d'un appareil CANopen

II.3.1 Architecture interne

L'architecture interne d'un appareil CANopen [12] se compose de trois parties logiques : l'interface de communication qui gère la communication via le bus CAN, le processus d'application qui fournit les fonctionnalités de l'appareil et le Dictionnaire Objets du CANopen qui se situe entre l'interface de communication et le processus d'application. Ceci est détaillé dans la figure II-6.

Cette architecture est basée sur un « profil de communication » qui précise les mécanismes de base de la communication. Ce profil décrit les types d'appareils les plus standards. Si nous examinons quelques exemples de profil de communication nous citons : les modules I/O analogiques et numériques, les lecteurs, les automates programmables et les codeurs. Dans la partie fonctionnalité de l'appareil, les paramètres des appareils standards sont définis ainsi que les différents types d'accès au traitement de données. Le Dictionnaire Objets contient des références pour tous les types de données utilisées et tous les paramètres de communication et d'application.

Nous allons décrire dans ce qui suit la partie Dictionnaire Objet. Nous allons réserver la troisième section de ce chapitre pour détailler l'interface de communication et les différents messages CANopen.

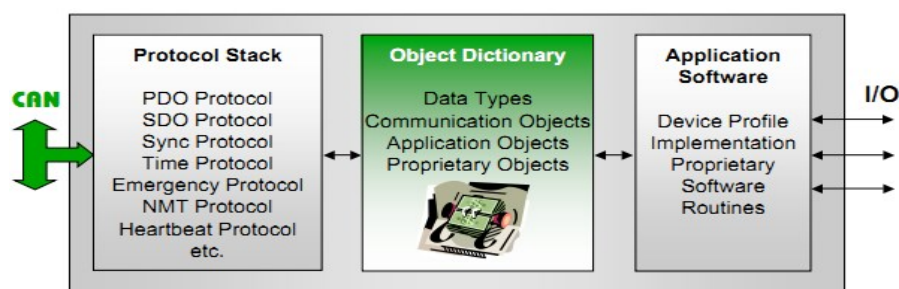


Figure II-6 : Architecture appareil CANopen

II.3.2 Objets et Dictionnaire Objets

Le Dictionnaire Objet est le cœur de tout dispositif CANopen, Il permet de décrire les fonctionnalités de l'appareil [13]. Chacune des données qui doivent être échangées entre l'interface de communication et de la partie de l'application du dispositif possède son adresse unique de 24 bits. Cette adresse est divisée en Index codé sur 16 bits et Sub-index sur 8 bits.

Un Dictionnaire Objets peut contenir jusqu'à 65536 entrées qui sont traitées par l'index. Il suit l'organisation du tableau suivant conformément à la norme CANopen.

Index (hex)	Type d'objets
0000	Réservé
0001-001F	Types de données statiques
0020-003F	Type de données complexes
0040-005F	Types de données complexes spécifiques au fabricant de l'équipement
0060-007F	Types de données statiques spécifiques au profil de l'équipement
0080-009F	Types de données complexes spécifiques au profil de l'équipement
00A0-0FFF	Réservé
1000-1FFF	Objets de communication
2000-5FFF	Objets spécifiques au fabricant
6000-9FFF	Objets spécifiques au profil d'équipement
A000-FFFF	Réservé

Tableau II-1 : Dictionnaire Objet

Remarque : Seuls les éléments grisés sont accessibles dans le Dictionnaire Objet par les nœuds connectés au réseau.

L'accès au Dictionnaire Objet se fait toujours avec l'utilisation de l'index seulement. Dans le cas d'une variable simple, on accède directement à la valeur de cette variable. Pour les entrées du Dictionnaire Objets complexes comme les tableaux ou les enregistrements à plusieurs champs de données, on utilise le champ Sub-index d'une structure de donnée pointée principalement par l'index.

Nous allons prendre un exemple sur un seul canal d'interface RS-232. Dans cette interface il peut exister une structure de données qui définit les paramètres de communication de ce module, elle peut contenir différents champs comme il est indiqué dans la structure en C ci-dessous de l'RS232_T.

Structure en C :

```
typedef struct {
    UNSIGNED 8 Nombre d'entré
    UNSIGNED 16 Débit
    UNSIGNED 8 Nombre de bit de données
    UNSIGNED 8 Nombre de bits d'arrêt
    UNSIGNED 8 parités
} RS232_T
```

Cette structure est décrite dans l'index 6092h, on peut utiliser le Sub-index pour accéder à ces différents champs comme il est indiqué dans le tableau II-2.

Index	Sub-Index	Variable accordée	Type de données
6092	0	Nombre d'entré	UNSIGNED 8
6092	1	Débit	UNSIGNED 16
6092	2	Nombre de bit de données	UNSIGNED 8
6092	3	Nombre de bits d'arrêt	UNSIGNED 8
6092	4	Parités	UNSIGNED 8

Tableau II-2 : Exemple d'Index channel RS-232

Remarque : Pour les types de données utilisées nous avons défini UNSIGNED 8 (unsigned char), UNSIGNED16 (unsigned int) et UNSIGNED32 (unsigned long int).

II.4 Interfaces de communication

Pour faire la communication organisée sur le bus CAN [14], chaque trame CANopen commence par le champ d'arbitrage. Nous avons vu que la priorité est attribuée selon la valeur du champ d'arbitrage sur 11 bits. Selon la priorité des objets CANopen, chaque objet possède un code qui lui identifie un COB_ID (Communication Object Identifier) sur 4 bit et les 7 suivants correspondent à l'identifiant des nœuds ID (Node_ID) pour faire la différence entre un maximum de 127 nœuds.

Les systèmes qui implémentent le protocole CANopen possèdent deux mécanismes de transfert de données SDO et PDO et d'autres protocoles de gestion de réseau : NMT (pour la gestion globale du réseau), Emergency (pour la signalisation d'erreurs), SYNC (pour la synchronisation) et Time stamp (synchronisation à haute résolution), ainsi qu'un protocole de

contrôle de l'état des nœuds (Heartbeat). Nous allons décrire dans ce qui suit ces différents protocoles.

II.4.1 Gestion de réseau

II.4.1.1 Protocole NMT

Tous les appareils CANopen doivent supporter le protocole Network Management (NMT) qui définit leur comportement de communication. La gestion du réseau (NMT) suit le modèle de communication maître/esclave. Dans un réseau CANopen il y a un seul maître NMT actif et les autres nœuds sont des NMT Esclaves.

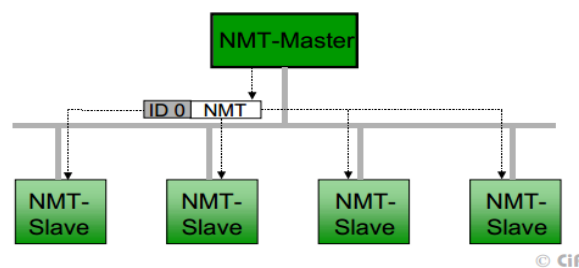


Figure II-7: Network Management (NMT)

Le but de ce protocole est de contrôler l'état des périphériques esclaves NMT dans le réseau CANopen. Les esclaves NMT fonctionnent selon 4 états différents qui conditionnent les messages qu'ils peuvent envoyer/recevoir. Ces différents états sont organisés dans le diagramme de la figure II-8.

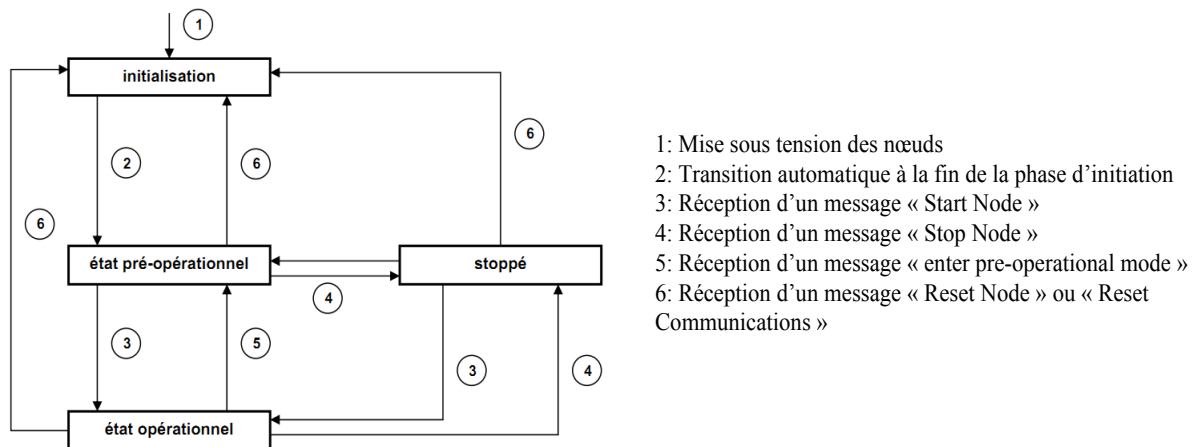


Figure II-8 : Diagramme d'état des esclaves NMT

Comme il est indiqué dans la figure II-9, les messages NMT sont constitués de la même façon, une seule trame de données CAN avec une longueur de données (DLC) de deux octets. Le premier octet contient le spécificateur de commande CS et le second contient

l'identifiant du nœud (Node_ID) qui doit exécuter la commande. L'identifiant du CAN (COB_ID) du message NMT est 0h.

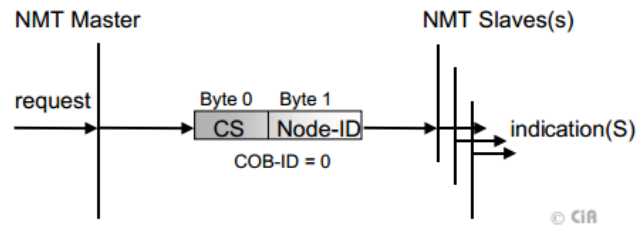


Figure II-9 : Message NMT

Le message NMT fournit cinq services qui peuvent être distingués par le champ CS, le tableau II-3 résume les états des variables du message NMT.

Paramètre	Désignation
Node_ID	Il doit être compris entre 0h et 7Fh 0 : faire une diffusion d'état à tous les nœuds
CS (Command Specifier)	1 : Initialisation du nœud 2 : Arrêt du nœud 128 : Etat pré-opérationnel 129 : Réinitialiser le Nœud 130 : Réinitialiser la communication

Tableau II-3 : Variables du message NMT

II.4.1.2 Synchronisation

Le protocole SYNC est diffusé périodiquement sur le réseau CAN par un nœud producteur SYNC, c'est l'horloge de base du réseau. Il utilise le modèle de communication Producteur/Consommateur. Le message SYNC cycliquement transmis, indique aux consommateurs de démarrer leur comportement spécifique à l'application qui est lié à la réception du message SYNC. Il y a un producteur de SYNC et plusieurs consommateurs de synchronisation comme c'est indiqué par la figure II-11.

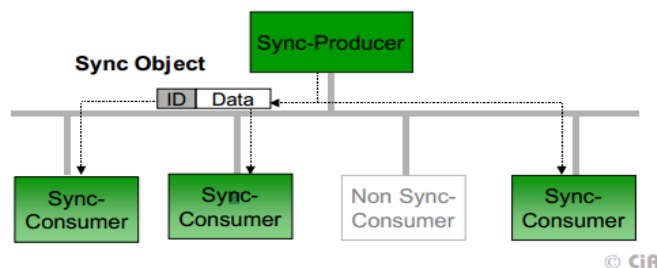


Figure II-10 : Synchronisation Object

Le service SYNC ne nécessite aucun message de confirmation de la part des nœuds consommateurs. Comme il est indiqué dans la figure II-11, le message SYNC ne contient aucune donnée (DLC=0). Les nœuds consommateurs du SYNC sont activés à l'entrée du COB_ID (80h), il peut être lu à l'index du Dictionnaire Objet 1005h (Sub-Index 00h).

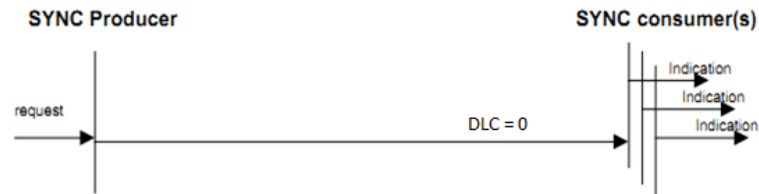


Figure II-11 : Message SYNC

Le temps d'une période est défini par le paramètre standard « communication cycle period » (1006h) et le paramètre « synchronous window length » (1007h) qui sont spécifiés dans le Dictionnaire Objet (voir figure II-12). Ils peuvent être configurés et envoyés à l'application pendant le démarrage.

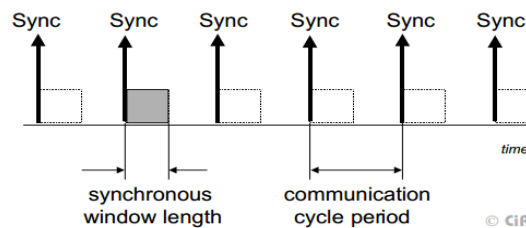


Figure II-12 : SYNC Time Définition

II.4.1.3 Time Stamp

L'objet Time Stamp représente le temps absolu en microsecondes après minuit et le nombre de jours depuis le 1^{er} Janvier 1984.

Certaines applications qui sont critiques au temps, surtout dans les grands réseaux avec des débits de transmission réduits, nécessitent une synchronisation très précise, il est nécessaire pour synchroniser les horloges locaux avec une précision de l'ordre de microseconde.

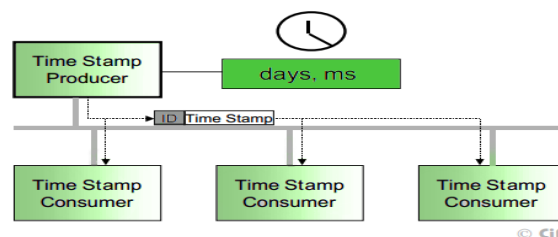


Figure II-13 : Time Stamp

La haute résolution Time Stamp est codée comme UNSIGNED 32 avec une résolution d'une microseconde, ce qui signifie que le temps redémarre toutes les 72 minutes, cette spécification est configuré dans l'objet 1013h.

Le message Time Stamp est associée à une seule trame CAN codée sur 48bit (DLC=6 octets), cette trame est identifié par un COB_ID=100h, comme le montre la figure II-14.

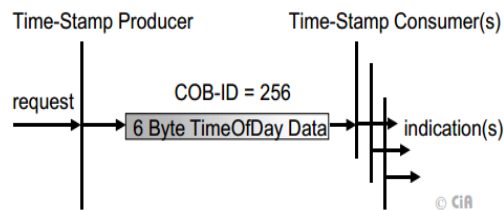


Figure II-14 : Trame Time Stamp

II.4.1.4 Les messages d'urgence

Les messages d'urgence (Emergency) sont déclenchés par la présence d'un dispositif d'erreur fatale. Un message d'urgence permet à un nœud d'envoyer un message contenant le code d'erreur correspondant au problème auquel il est confronté aux autres appareils avec une priorité élevée. Ce message d'erreur permet de déclencher des alertes de type d'interruption.

Le message d'urgence est diffusé. Un ou plusieurs dispositifs producteurs peuvent transmettre un message d'urgence et un seul dispositif consomme cette information.

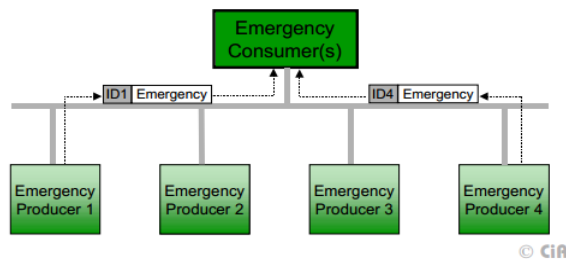


Figure II-15 : Emergency Service

Par défaut, un dispositif qui prend en charge la fonctionnalité de producteur d'urgence affecte le COB_ID (80 h + Node_ID), à savoir que le Node_ID est l'identificateur du nœud qui avec un maximum de dispositifs connectés 7Fh (127 nœuds).

Les messages d'urgence sont émis sur le bus mais ne requièrent pas de confirmation de la part des consommateurs. La figure II-16 décrit les 8 bits de données envoyées dans la trame Emergency qui sont répartis de la manière suivante : les deux premiers octets contiennent le

code d'erreur. Le second contient le registre d'erreur. Les cinq derniers octets constituent le champ d'erreur spécifique au constructeur.

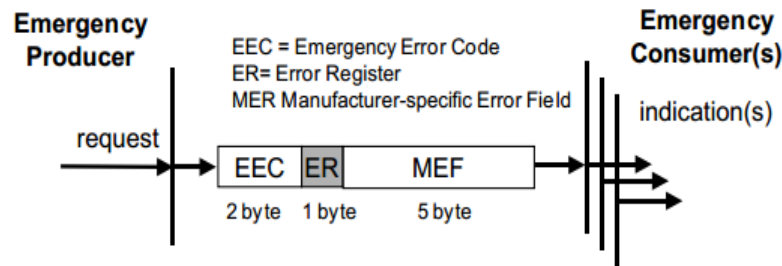


Figure II-16 : Trame Emergency

Le tableau II-4 décrit quelques codes d'erreur avec leur description.

Code de l'Erreur (hex)	Description de l'Erreur
0	Erreur acquittée
10	Erreur générique
3200	Défaut de tension convertisseur
8120	Erreur de communication CAN
8130	Erreur de Heartbeat

Tableau II-4: Emergency Error Code

II.4.2 Protocole de contrôle de l'état des nœuds Heartbeat

Le protocole Heartbeat définit un service de contrôle d'état d'un nœud, il peut aussi contrôler le retard d'envoi dû à une défaillance d'un nœud en générant un « Heartbeat evant ».

Les messages envoyés par le Heartbeat Producer sont associés à une seule trame CAN avec une longueur de données d'un octet (DLC=1), l'identifiant du message Heartbeat est (COB_ID=700h).

Le producteur Heartbeat transmet un message d'une façon cyclique avec une fréquence choisie appelé « Heartbeat Producer Time », spécifié dans l'index (1017h). Un ou plusieurs Consommateur Heartbeat peuvent recevoir le message.

A la réception, le consommateur Heartbeat enregistre la valeur de la pulsation dans le « Heartbeat Consumer Time » spécifié dans l'index (1016h). Si le signal n'est pas reçu dans le « Heartbeat Consumer Time » un événement Heartbeat sera généré.

La figure II-17 décrit le trafic des messages Heartbeat.

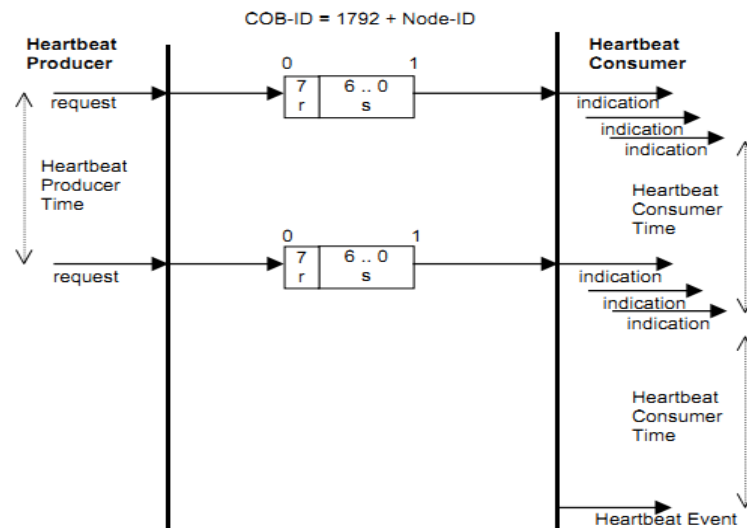


Figure II-17 : Message Heartbeat

Le tableau II-5 résume l'état des variables du message Heartbeat.

Paramètre	Désignation
r (réservé)	Toujours à 0
s (état du producteur de Heartbeat)	0 : redémarrage 4 : arrêté 5 : opérationnel 127 : pré-opérationnel

Tableau II-5 : Variables du message Heartbeat

II.4.3 Messages d'accès au dictionnaire d'objets

CANopen distingue deux mécanismes basiques spécialisés au transfert de données. Le SDO (Service Data Object) qui permet l'accès aux entrées du le Dictionnaire Objet et le PDO (Process Data Object) qui permet l'échange des données de petites tailles, à grande vitesse, entre les nœuds connectés au réseau.

II.4.3.1 Message SDO

Le message SDO (Service Data Object) suit le modèle de communication Client/Serveur. Il permet l'accès au Dictionnaire Objet, d'un appareil CANopen. Le Client contrôle la donnée qui doit être transmise à travers le multiplexeur (Index et Sub-Index du Dictionnaire Objet). Un SDO se compose de deux trames de données CAN. A la transmission, la trame contient un identificateur COB_ID qui sera égale $0x600 + \text{Node_ID}$, et à la réception le COB_ID sera égale à $0x580 + \text{Node_ID}$.

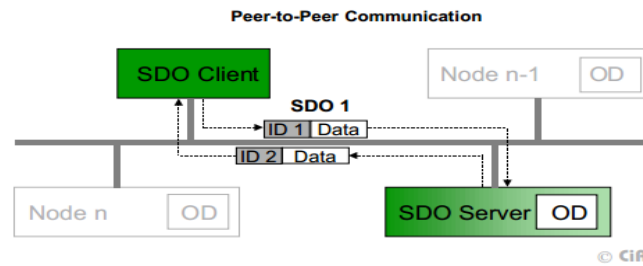


Figure II-18: Communication SDO

II.4.3.1.1 Les services SDO

Le protocole SDO définit deux services [15], SDO Download et SDO Upload. Un SDO Download est la transmission de données du Client vers le Serveur, et un SDO Upload consiste à transmettre les données du Serveur vers le Client. Le dispositif qui accède au Dictionnaire Objet de l'autre appareil est le client SDO.

Un appareil CANopen peut soutenir deux types de transfert : un transfert accéléré (Expedited transfert) et un transfert normal (Segmented transfert), le tableau III-3 décrit brièvement ces types de transfert.

Type de transfert	Longueur de données	Description
transfert accéléré (Expedited transfert)	1 – 4 octet	Les données sont déjà transmises lors de l'initiation du transfert de données. Il est confirmé par un message de réponse.
transfert segmenté (normal transfert)	1 - >64 kilo octet	Seule la longueur du paquet de données est transmise lors de l'initiation du transfert de données. Les données sont transmises en segments de 7 octets de données et un octet de protocole chacun. Chaque segment est confirmé par un message de réponse.

Tableau II-6 : Description des types de transfert SDO

Pour faire un transfert accéléré, le Client envoie une demande d'initialisation du SDO Download ou Upload, suivie d'une réponse de confirmation du Serveur indiquant la réussite du Download/Upload accéléré. Les trames échangées entre le Client et le Serveur sont différents selon le type d'opération Download ou Upload. La figure II-19 décrit le protocole de transfert de données SDO en mode accéléré. Le champ CS (Command Specifier) est codé sur un octet et indique le type d'opération Download ou Upload et le mode de transfert. Le champ MUX est codé sur 3 octets et indique l'Index et le Sub-Index des données dans le

Dictionnaire Objet. Le dernier champ DATA est codé sur 4 octets et contient les données qui vont être envoyées. En cas de confirmation de Download ou de demande d'Upload, ce champ est non utilisé.

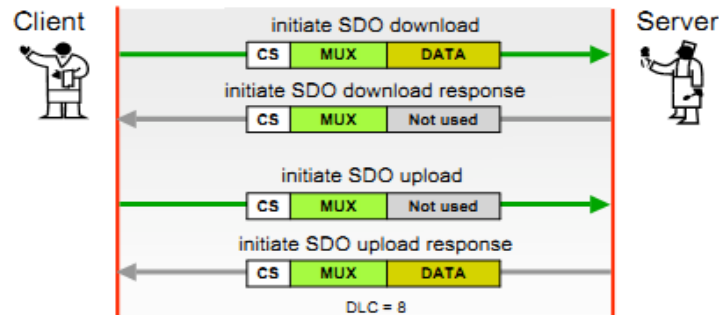


Figure II-19 : Protocol Download/Upload SDO en mode accéléré

Le champ CS utilise la même longueur de donnée pour la demande Download SDO et la réponse Upload SDO, ainsi que pour la réponse Download SDO et la demande Upload SDO. La figure II-20 indique que les champs de l'octet CS sont similaires pour les modes d'opération Download et Upload.

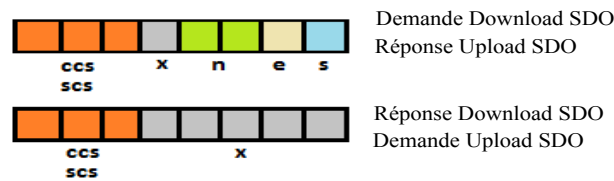


Figure II-20 : Description du champ CS en mode accéléré

En mode de transfert accéléré la variable e est égal à 1. Le tableau II-7 décrit les différents champs et variables de l'octet CS.

Paramètre	Désignation	Nombre de bits de codage
CSS SCS	Client command specifier Server command specifier	3 premiers bits
x	Non utilisé, toujours à 0	- 1 bit pour SDO download et SDO upload response - 5bits pour SDO download response et SDO Upload
n	Nombre d'octets dans le champ DATA qui ne contiennent pas de données. valable si $e = 1$ et $s = 1$, sinon il est égal à 0	2 bits
e	Type de transfert 0 : transfert normale (Normal transfer) 1 : transfert rapide (Expedited transfer)	1 bit
s	Indicateur de taille dans le champ DATA 0 : donnée n'est pas indiquée 1 : donnée est indiquée	1 bit

Tableau II-7 : Variables du de l'octet CS

Lors du transfert normal de données, la première trame échangée entre le client et le serveur contient la taille des données à envoyer, dans le champ DATA. On retrouve le même champ CS lors du transfert accéléré, sauf que la variable e devient égal à 0. La figure II-21 décrit le protocole de transfert de données SDO en mode normal pour une opération de type Download. Une fois le processus de transfert de données est déclenché, le client envoie les trames SDO segmentées qui contiennent un champ CS sur 1 octet et un champ DATA sur 7 octets. Le serveur renvoie une trame de confirmation indiquant le bon déroulement du processus et qui contient seulement le champ CS.

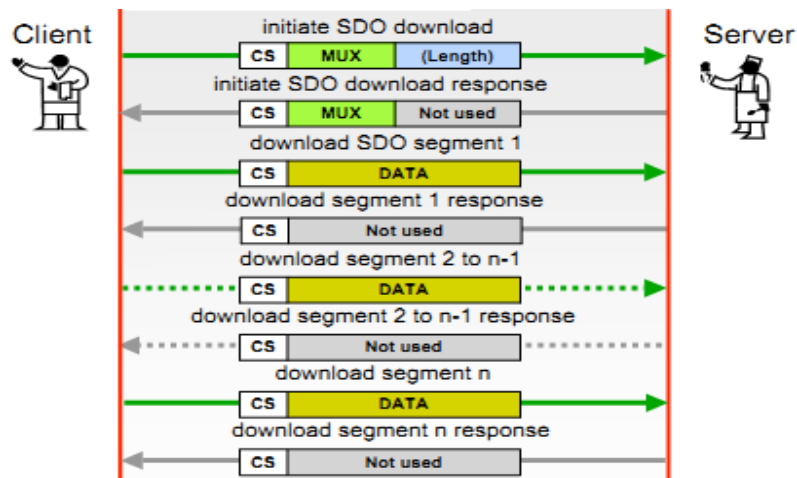


Figure II-21 : Protocol Download SDO segmenté en mode normal

En mode de transfert normal, le champ e est mis à 1. Le bit de bascule t est mis à 0 pour le premier transfert et il est changé à chaque nouvel envoi. Le bit c est mis à 1 pour indiquer qu'il y a encore de données à transférer. La figure II-22 montre que les champs de l'octet CS admettent la même taille des données, pour les modes d'opération Download et Upload.

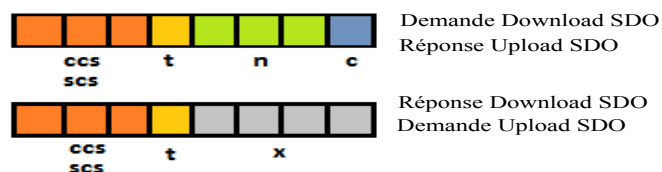


Figure II-22 : Description du champ Command Specifier (CS) en mode normal

En cas d'échec de transfert Download ou Upload, un message de transfert Abort SDO est envoyé. Ceci est indiqué par la figure II-23.

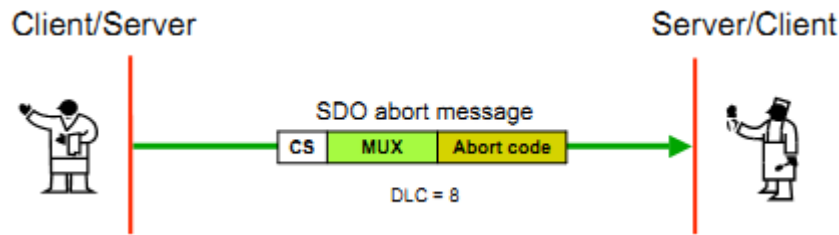


Figure II-23 : Protocol Abort SDO

Pour l'Abort code qui est envoyé dans le champ Data, il y a plusieurs type d'erreurs, on cite quelques exemples dans le tableau II-8.

Code d'erreur	Description
6010001h	Accès non supporté pour cet objet
6020001h	L'objet demandé n'existe pas dans le Dictionnaire Objet
6040042h	La taille des objets PDO à inclure dans le PDO excède la taille max
6010001h	Donnée de taille trop importante pour l'objet considéré

Tableau II-8 : SDO Abort code

II.4.3.1.2 SDO dans le Dictionnaire Objet

Le COB_ID des messages respectifs du transfert SDO du client vers le serveur et du serveur vers le client peut être défini dans le Dictionnaire Objet. Jusqu'à 127 serveurs SDO peuvent être mis en place dans le Dictionnaire Objet suivant une plage d'adresse de 0x1200 à 0x127F.

De même, les connexions client SDO peuvent être configurées avec des variables de l'adresse 0x1280 à 0x12FF. Les paramètres des SDO sont regroupés dans le tableau II-9.

Paramètre Serveur SDO				
1200h	Enregistrer	Paramètre 1 ^{er} Serveur SDO	SDOParameter	rw
127Fh	Enregistrer	Paramètre 128 ^{ème} Serveur SDO	SDOParameter	rw
Paramètre Client SDO				
1280h	Enregistrer	Paramètre 1 ^{er} Client SDO	SDOParameter	rw
12FFh	Enregistrer	Paramètre 128 ^{ème} Client SDO	SDOParameter	rw

Tableau II-9 : Les objets des SDO

Remarque: les paramètres du Serveur SDO et du Client SDO sont enregistrés dans l'index 22h.

II.4.3.2 Message PDO

Les PDO (Process Data Object) sont utilisés dans le CANopen pour la diffusion d'information de contrôle, ils manipulent les données du Dictionnaire Objet et génèrent les informations nécessaires pour le fonctionnement d'une application spécifique. Les objets PDO suivent l'architecture Producteur/Consommateur. En effet, la donnée est émise d'un

nœud producteur Transmit-PDO (T_PDO) vers un ou plusieurs consommateurs Receive-PDO (R_PDO), et ceci sans demande de confirmation

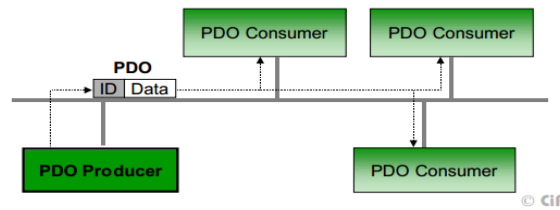


Figure II-24: Communication PDO

Un message PDO se compose d'une seule trame CAN, il permet de communiquer jusqu'à 8 octets de données (en maximum DLC=8). Dans le protocole CANopen, il y a des identifiants pour quatre T_PDO et pour quatre R_PDO disponibles, l'identifiant (COB_ID) du premier T_PDO1 est dans l'intervalle [181h -1FFh] et l'identifiant du premier R_PDO1 est dans l'intervalle [201 h- 280 h].

II.4.3.2.1 Les services PDO

Dans un traitement PDO, Il y a deux services : écrire et lire PDO. L'écriture PDO est associée à une seule trame de données, alors que la lecture PDO se réalise par une demande venue du consommateur. Ensuite, le producteur lui répond par une trame de données (Data Frame), comme il est indiqué par la figure II-19.

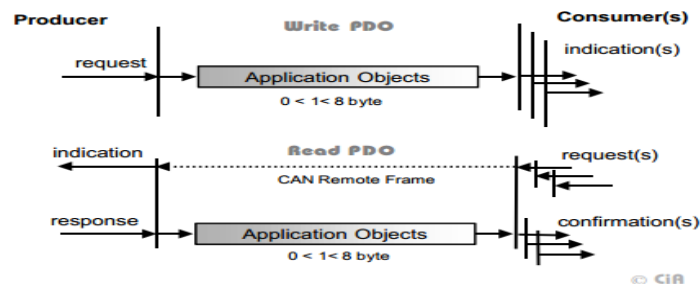


Figure II-25 : Services PDO

Le déclenchement d'un message PDO est dû à la réception de l'un de ces trois événements:

- La transmission de messages est déclenchée par l'apparition d'un événement spécifique, ou un signal déclenché par une minuterie écoulé (Timer).
- La transmission de PDO asynchrone peut être lancée lors de la réception d'un Remote Frame initiée par un autre dispositif.
- Les PDO synchrones sont déclenchées par l'expiration d'une donnée périodique SYNC.

La figure II-20 résume les 3 événements qui déclenchent l'envoi d'un message PDO.

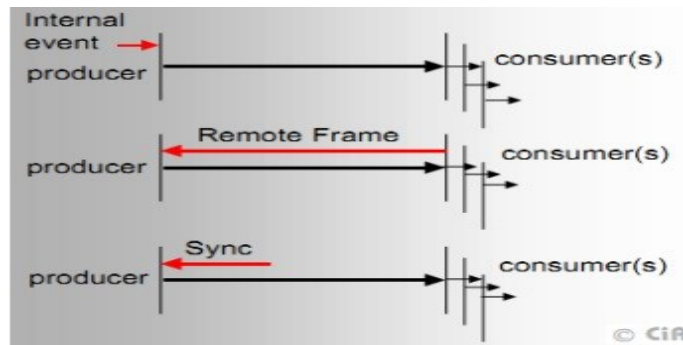


Figure II-26 : déclenchement pdo

Il y a deux paramètres qui permettent de décrire un PDO:

- Le paramètre PDO Mapping qui définit les objets d'applications qui vont être transmise dans des PDO.
- Le paramètre PDO Communication décrit les possibilités de communication PDO.

II.4.3.2.2 Le paramètre PDO Mapping

Le PDO mapping définit les objets d'application qui seront transmis dans un message PDO. Il décrit la séquence et la longueur des objets d'application mappés. La cartographie des objets d'application est décrite dans les entrées du Dictionnaire Objet CANopen pour chaque PDO. En effet pour envoyer une trame mappée, nous devons spécifier les Mux (Index et Sub-Index) des données à mapper, ensuite, nous devons les chercher dans le Dictionnaire Objet, et enfin nous construisons la trame T_PDO1 en y mettant les données trouvées. Cette trame peut être envoyée directement aux consommateurs ou sauvegardée dans le Dictionnaire Objet pour un envoi ultérieur.

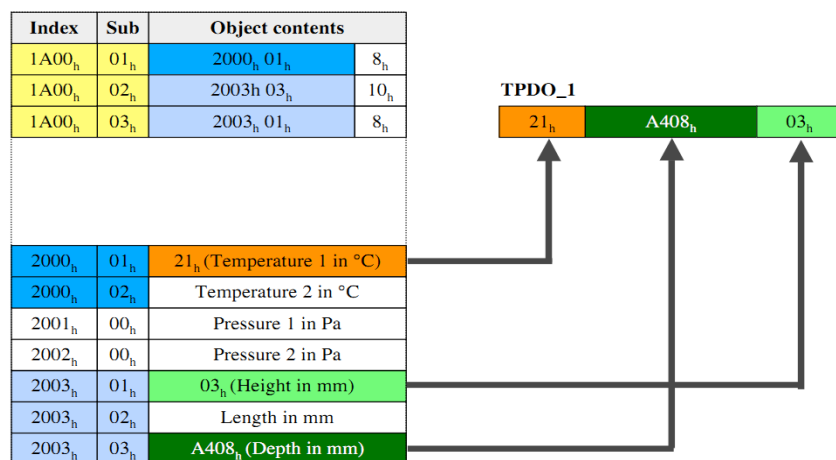


Figure II-27 : PDO Mapping

Le tableau II-9 indique le paramètre de mappage PDO qui est enregistré dans l'Index 21h. Cet Index (21h) peut supporter jusqu'à 64 objets mappés.

Index	Sub-Index	Description	Type de données
1600h	0h	Nombre de donné mappé	UNSIGNED 8
à 17FFh	1h	1 ^{er} objet	UNSIGNED 32
1A00h	2h	2 ^{ème} objet	UNSIGNED 32
à 1BFFh
	40h	64 ^{ème} objet	UNSIGNED 32

Tableau II-10 : Les paramètres du PDO mapping

II.4.3.2.3 Le paramètre PDO Communication

Le paramètre de communication PDO décrit les possibilités de communication des messages PDO. Les paramètres de communication PDO décrite par le tableau II-10 sont enregistrés dans l'Index 20h du Dictionnaire Objet.

Index	Sub-Index	Description	Type de données
1400h	0h	Nombre d'entrée	UNSIGNED 8
à 15FFh	1h	COB_ID	UNSIGNED 32
1800h	2h	Type de transmission	UNSIGNED 8
à 19FFh	3h	Inhibit time	UNSIGNED 16
	4h	réservé	UNSIGNED 8
	5h	event timer	UNSIGNED 16

Tableau II-11 : Les paramètres de communication PDO

II.4.3.2.4 PDO dans le Dictionnaire Objet

Les objets PDO dans le Dictionnaire Objet sont décrits dans la figure II-21, le contenu des PDO doit être défini lors de l'initialisation. Le PDO Mapping est défini par l'Index 1600h pour la première entrée R_PDO1, et 1A00h pour la première entrée T_PDO1. Pour la communication des paramètres PDO, elle est défini par l'index 1400h pour le premier R_PDO et 1800h pour le premier T_PDO.

Recevoir Paramètre PDO Communication				
1400h	Enregistrer	1 ^{er} paramètre recevoir PDO	PDOCommPar	rw
15FFh	Enregistrer	512 ^{ème} paramètre recevoir PDO	PDOCommPar	rw
Recevoir Paramètre PDO Mapping				
1600h		Recevoir 1 ^{er} PDO Mapping	PDO Mapping	rw
17FFh		Recevoir 512 ^{ème} PDO Mapping	PDO Mapping	rw
Transmettre Paramètre PDO Communication				
1800h	Enregistrer	1 ^{er} paramètre transmettre PDO	PDOCommPar	rw
19FFh	Enregistrer	512 ^{ème} paramètre transmettre PDO	PDOCommPar	rw
Transmettre Paramètre PDO Mapping				
1A00h		Transmettre 1 ^{er} PDO Mapping	PDO Mapping	rw
1BFFh		Transmettre 512 ^{ème} PDO Mapping	PDO Mapping	rw

Tableau II-12 : Les objets des PDO

II.5 Conclusion

Nous avons présenté dans ce chapitre, le protocole CANopen. En effet, nous avons détaillé les mécanismes de transfert de données, ainsi que les différentes fonctions spéciales et de base. Nous présenterons, dans le chapitre suivant, les étapes nécessaires à l'implémentation des différentes fonctionnalités du protocole CANopen.

Chapitre III

Implémentation du protocole CANopen sur DSP

III.1 Introduction

L'objectif de ce chapitre consiste à la mise en œuvre du protocole CANopen sur une plateforme DSP.

Dans ce chapitre, nous présenterons les différents organigrammes détaillant le fonctionnement du CANopen. Ensuite, nous détaillerons les environnements de développement hardware et software de notre application. Le protocole est implémenté sur un DSP du type Blackfin BF548 parce que ce processeur intègre déjà un bus de communication CAN. Nous avons utilisé le visual DSP++ pour le codage du protocole. La dernière partie du chapitre sera consacrée aux tests de validation du code et l'analyse des performances d'implémentation.

III.2 Organigrammes descriptifs du protocole CANopen

Dans cette section, nous allons décrire les différents algorithmes que nous avons développés pour la mise en œuvre du protocole CANopen.

III.2.1 Organigramme général du CANopen

L'organigramme de la figure III.1 décrit le trafic au niveau de la couche applicative CANopen. La lecture de la valeur du champ COB_ID identifie l'émetteur de la trame sur le Bus CAN. Ce champ permet de supporter des valeurs de 0000h jusqu'au 0700h. Selon la valeur du COB_ID, plusieurs actions sont menées :

- Si COB_ID = ID_NMT = 0000h, on fait un appel à la fonction **Protocole NMT**.
- Si COB_ID = ID_SYNC = 0080h et si la trame de données est vide, on envoie une trame de requête (Remote Frame) en mode de diffusion après un délai de temps égal à Sync-Counter généré par une fonction Timer.

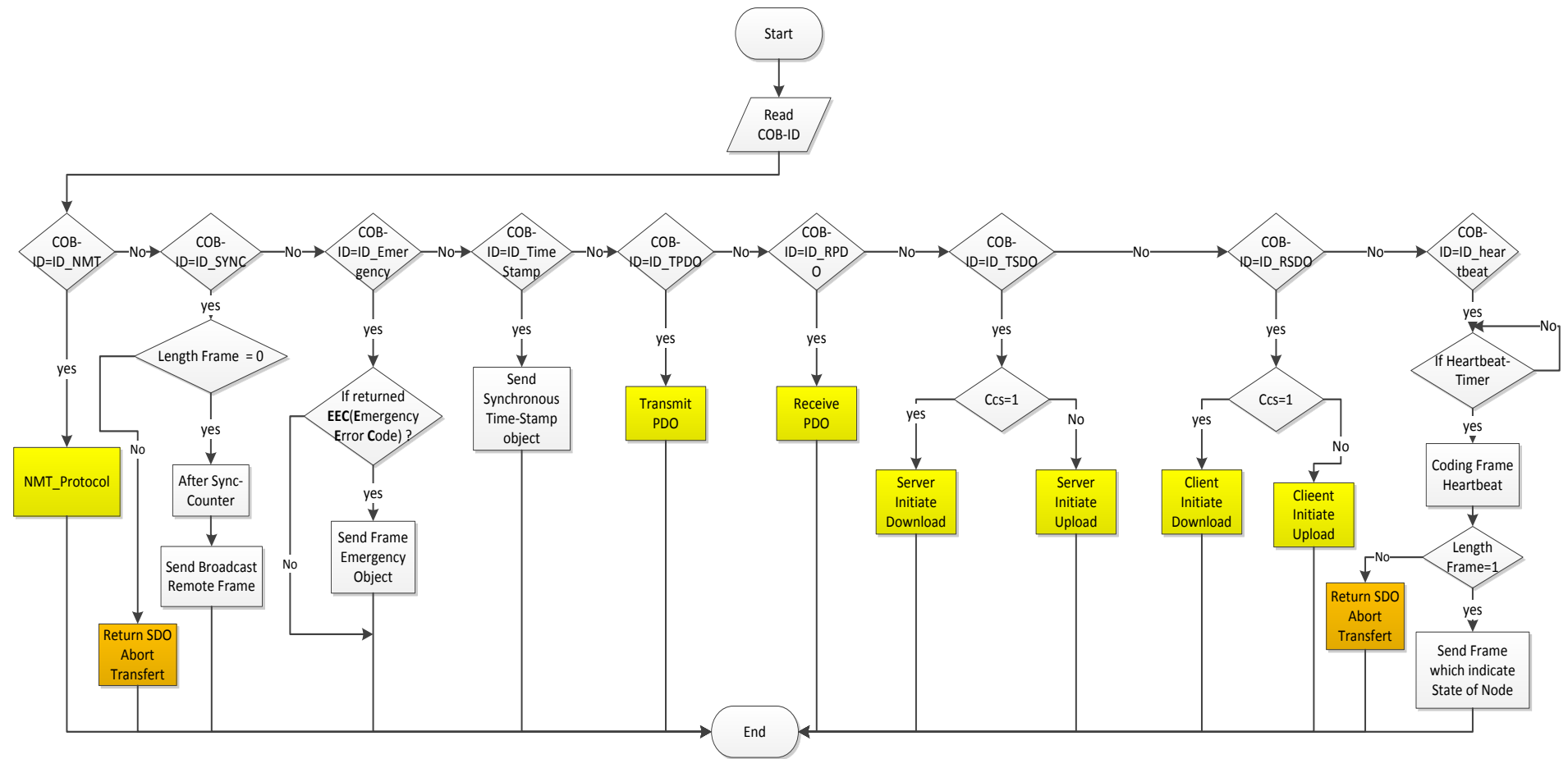


Figure III-1 : Organigramme général du CANopen

- Si COB_ID = ID_Emergency (« ID-Emergency >= 0080h » ET « ID-Emergency < 0080h+007Fh ») et si un nœud dans le réseau retourne un code prédéfini appelé EEC (Emergency Error Code), alors, on envoie un message d'urgence de taille 8 octets. Ce message contient 2 octets EEC (Emergency Error Code), 1 octet ER (Error Register) et 5 octets MER (Manufacturer specific Error field).
- Si COB_ID = ID_Time Stamp = 0100h, alors, on réserve 6 octets de données Time Stamp Object indiquant la date à partir du 1/1/1984. L'objectif de ce champ est de faire une synchronisation précise de l'ordre de microseconde nécessaire pour des applications critiques au temps.
- Si COB_ID = ID_TPDO (« ID_TPDO >= 0700h » ET « ID_TPDO < 0700h+007Fh »), alors, on fait un appel à la fonction **Transmettre PDO**.
- Si COB_ID = ID_RPDO (« ID_RPDO >= 0700h » ET « ID_RPDO < 0700h+007Fh »), alors, on fait un appel à la fonction **Recevoir PDO**.
- Si COB_ID = ID_TSDO (« ID_TSDO >= 0700h » ET « ID_TSDO < 0700h+007Fh »), alors, selon la valeur du champ CCS, on fait appel aux fonctions **Serveur Upload** ou **Serveur Download**.
- Si COB_ID = ID_RSDO (« ID_RSDO >= 0700h » ET « ID_RSDO < 0700h+007Fh »), alors, selon la valeur du champ CCS, on fait appel aux fonctions **Client Upload** ou **Client Download**.
- Si COB_ID = ID_Heartbeat (« ID_Heartbeat >= 0700h » ET « ID_Heartbeat < 0700h+007Fh »), alors, le compteur *Heartbeat-Timer* transmet un message d'une façon cyclique, et on envoie une donnée sur un octet permettant à un nœud d'indiquer aux autres son état (Arrêt, Opérationnel, Pré-Opérationnel, ...)

En cas d'erreur, on fait appel à la fonction **SDO Abort Transfert**. Nous allons décrire dans ce qui suit les organigrammes des fonctions **Protocole NMT**, **Transmettre PDO**, **Recevoir PDO**, **Serveur Upload**, **Serveur Download**, **Client Upload** et **Client Download**.

III.2.2 Organigramme du protocole NMT

Dans l'organigramme de la figure III.2, le champ Data de la trame CAN est représenté sur 2 Octets, un octet pour le champ CS et le deuxième pour l'identifiant du nœud (Node ID). Selon la valeur du champ CS, certaines actions sont effectuées :

Si CS=1, il s'agit d'un « Start Remote Node ».

Si CS=2, il s'agit d'un « Stop Remote Node ».

Si CS=128, il s'agit d'un «Enter Pre-Operational».

Si CS=129, il s'agit d'un «Reset Node».

Si CS=130, il s'agit d'un «Reset Communication».

Une fois le champ CS est défini, la valeur du Node_ID permettra de sélectionner le type d'envoi de données. Si Node_ID=0, les données sont diffusées à tous les nœuds connectés au réseau CAN, sinon il s'agit d'un envoi spécifique.

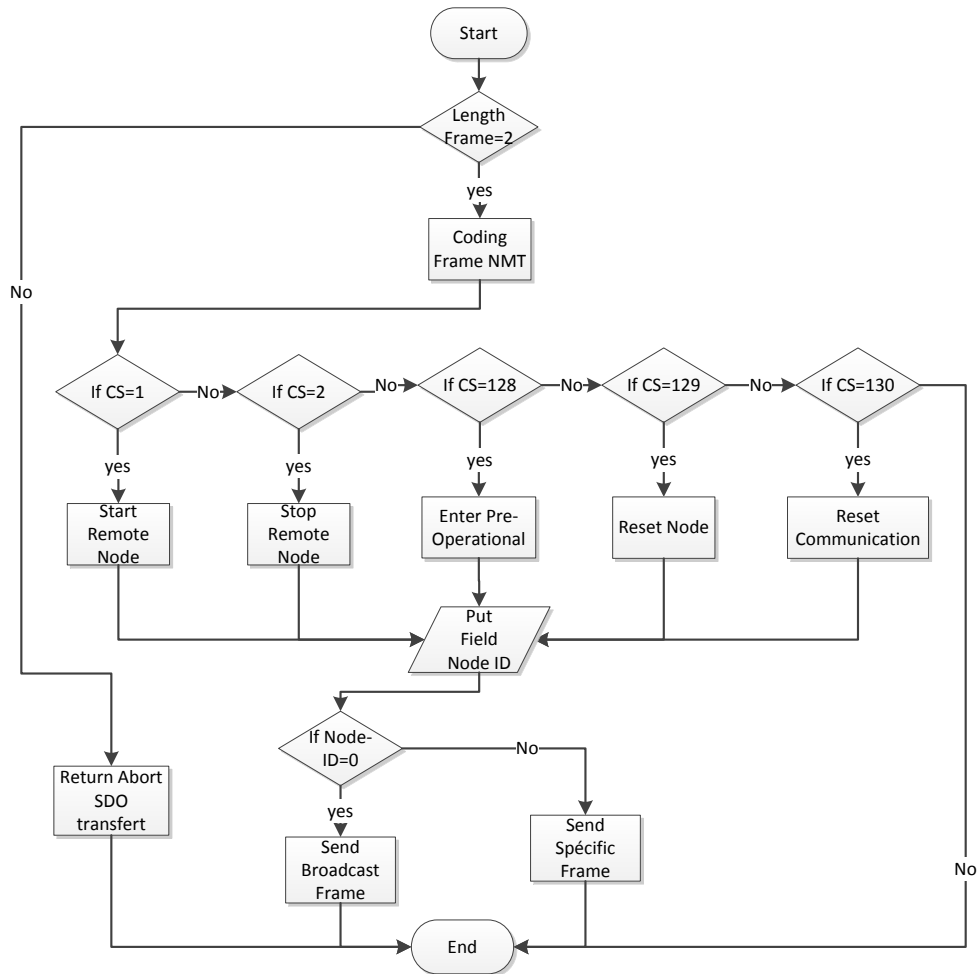


Figure III-2 : Organigramme NMT

III.2.3 Organigramme de la fonction Transmettre PDO

A chaque réception de la trame SYNC, on fait une diffusion des trames mappées. Si on reçoit une demande d'envoi du Consommateur, la trame mappée lui sera envoyée. En cas d'erreur, un Abort SDO est envoyé.

Remarque : la demande SYNC est plus prioritaire que la demande (Remote Frame) du Consommateur.

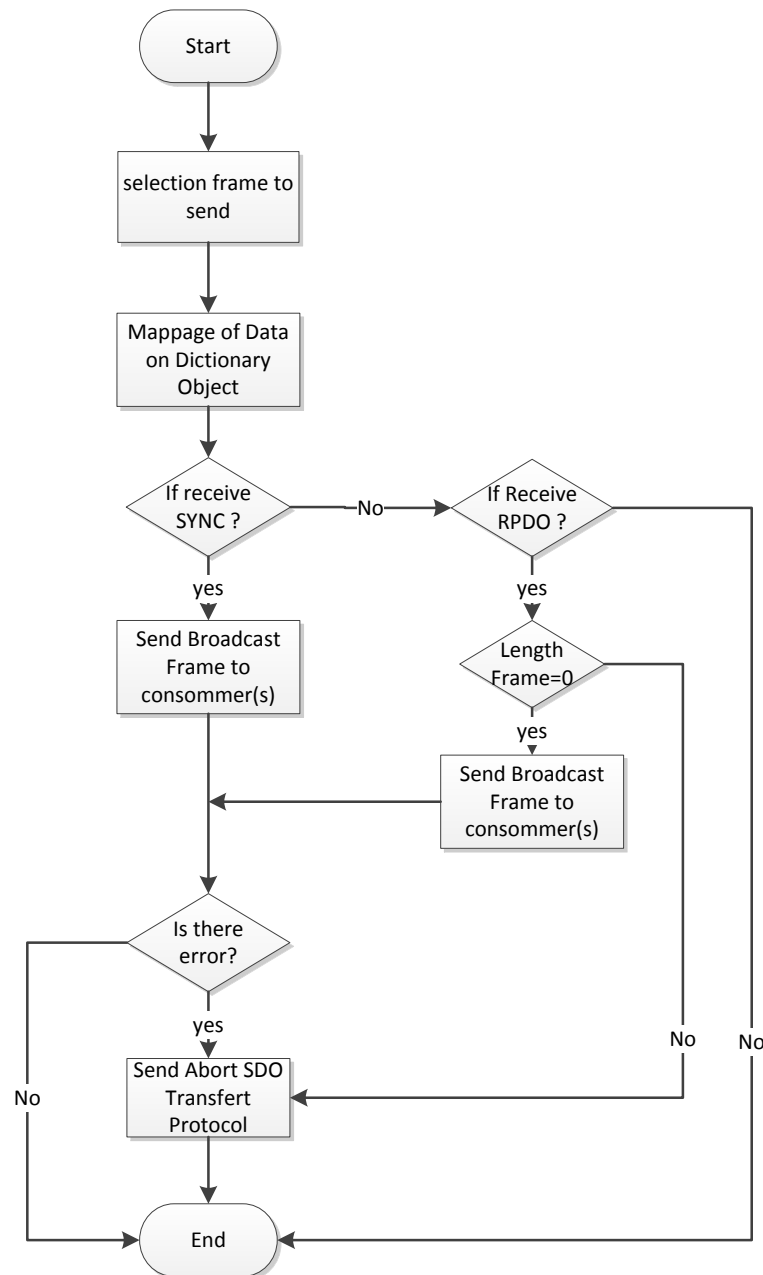


Figure III-3 : Organigramme Transmettre PDO

III.2.4 Organigramme de la fonction Recevoir PDO

Dans l'organigramme de la figure III-4, un premier test est effectué pour savoir si une trame SYNC est reçue et un deuxième pour savoir si le Consommateur a besoin d'une donnée (par l'envoi d'un Remote Frame). S'il y a une donnée parvenue du Producteur PDO, elle sera stockée dans le Dictionnaire Objet.

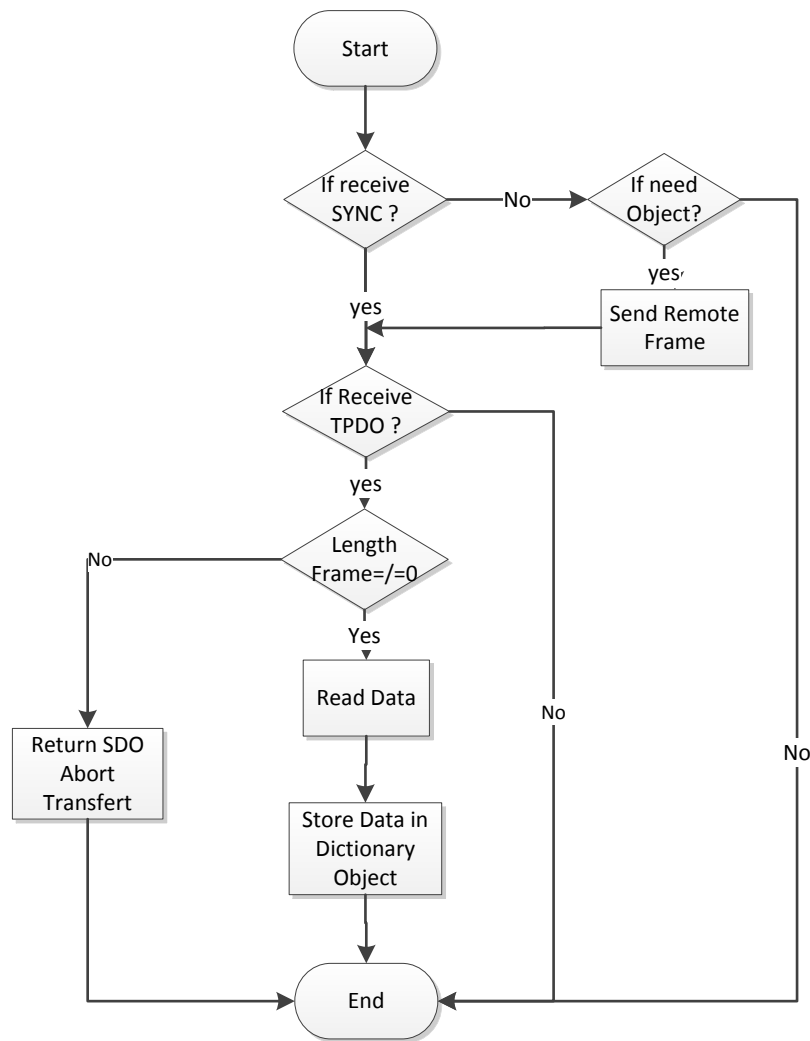


Figure III-4 : organigramme Recevoir PDO

III.2.5 Organigramme de la fonction Client Download SDO

Dans l'organigramme Client Download SDO, le client prépare une trame « Request Initiate SDO Download » qui contient la donnée à écrire, puis l'envoie vers le serveur. Ensuite, il reçoit le message de confirmation du serveur en vérifiant que le champ **SCS**=3, il lit la valeur du champ **e** et selon sa valeur, il fait le traitement comme suit :

- si **e**=0, le client fait un transfert accéléré (extented transfert).
- si **e**=1, le client fait un transfert normal (segmented transfert).

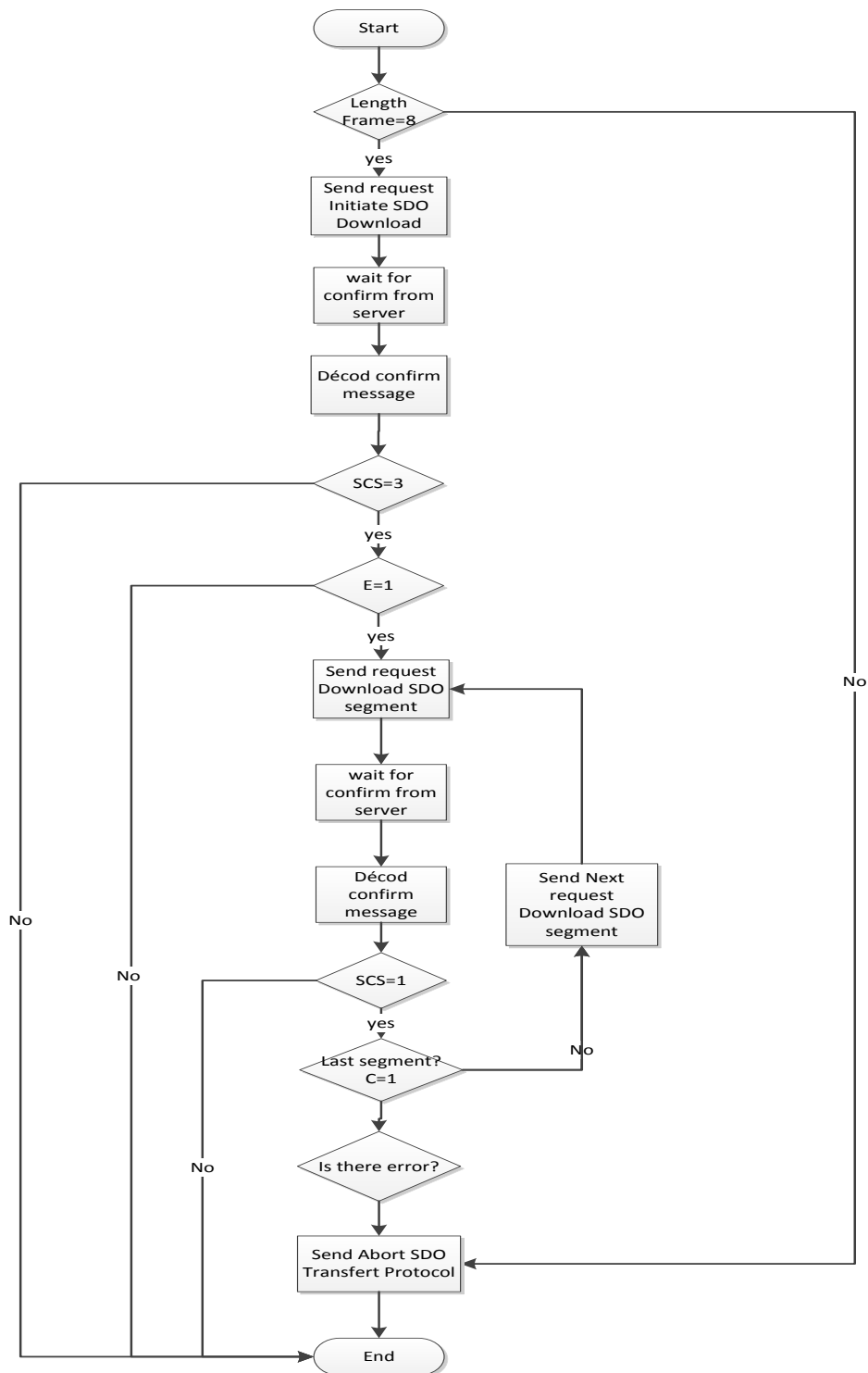


Figure III-5 : Organigramme Client Download SDO

III.2.6 Organigramme de la fonction Client upload SDO

Dans l'organigramme Client Upload SDO, le client prépare une demande « Request Initiate Upload SDO », puis, il l'envoie vers le serveur. Ensuite, il reçoit le message de confirmation qui contient les données à lire, il vérifie si le champ **SCS**=2, alors, il lit la valeur du champ **e**. Selon la valeur de **e**, il gère l'envoi de donnée en mode accéléré ou normal.

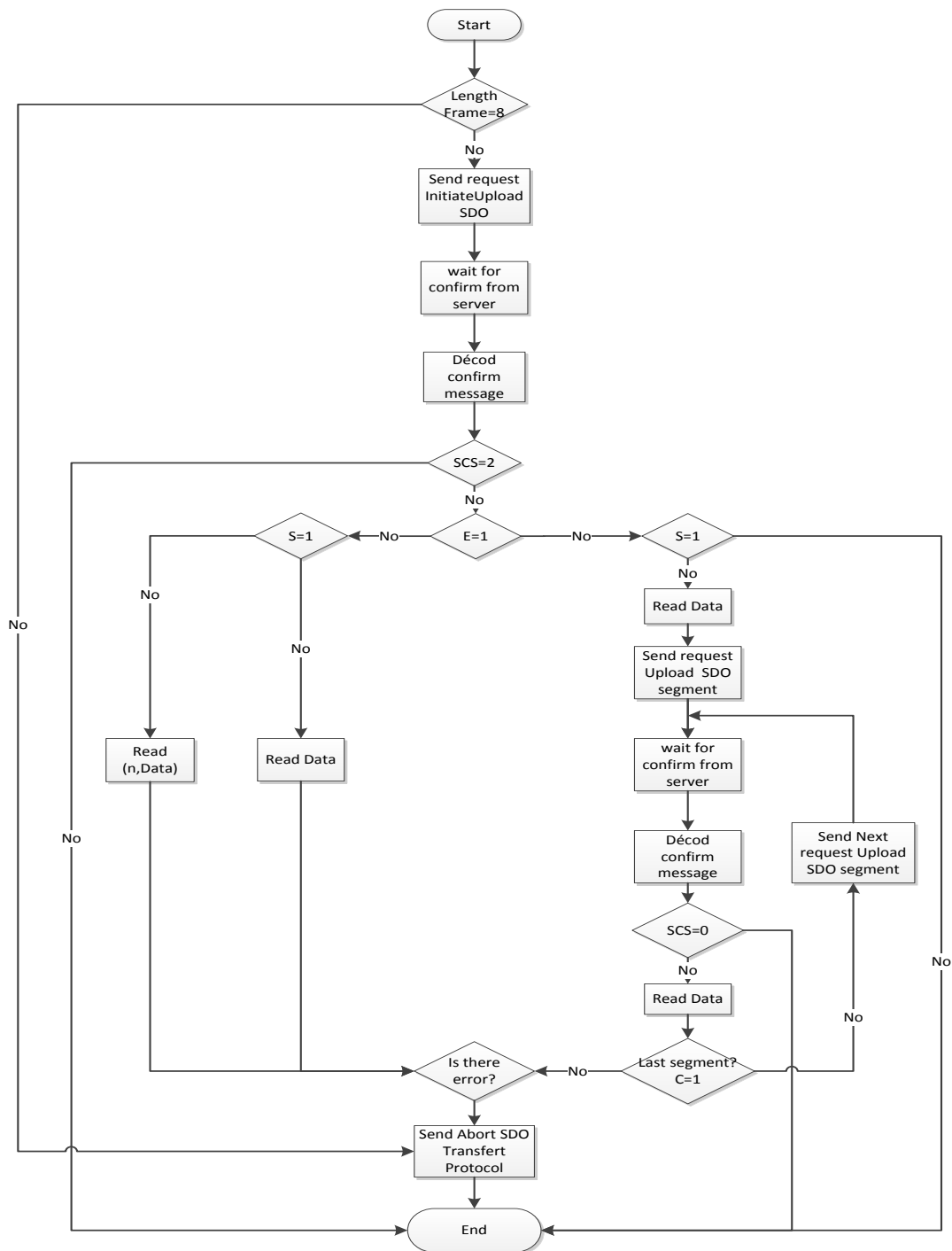


Figure III-6 : Organigramme Client Upload SDO

III.2.7 Organigramme de la fonction Serveur Download SDO

Dans cet organigramme, le serveur fait le décodage de la demande « Request Initiate SDO Download » envoyée par le client. En vérifiant que le champ **CCS=1**, il lit les bits *e* et *s* et selon leurs valeurs, différents traitements sont menés :

- si e=1 et s= 1 ou s=0, on procède au stockage accéléré des données dans le Dictionnaire Objet (extended transfert).

- si $e=0$ et $s=1$, on procède au stockage normal des données dans le Dictionnaire Objet (segmented transfert).
- si $e=0$ et $s=0$, le champ de données est vide, il est réservé pour une utilisation ultérieure.

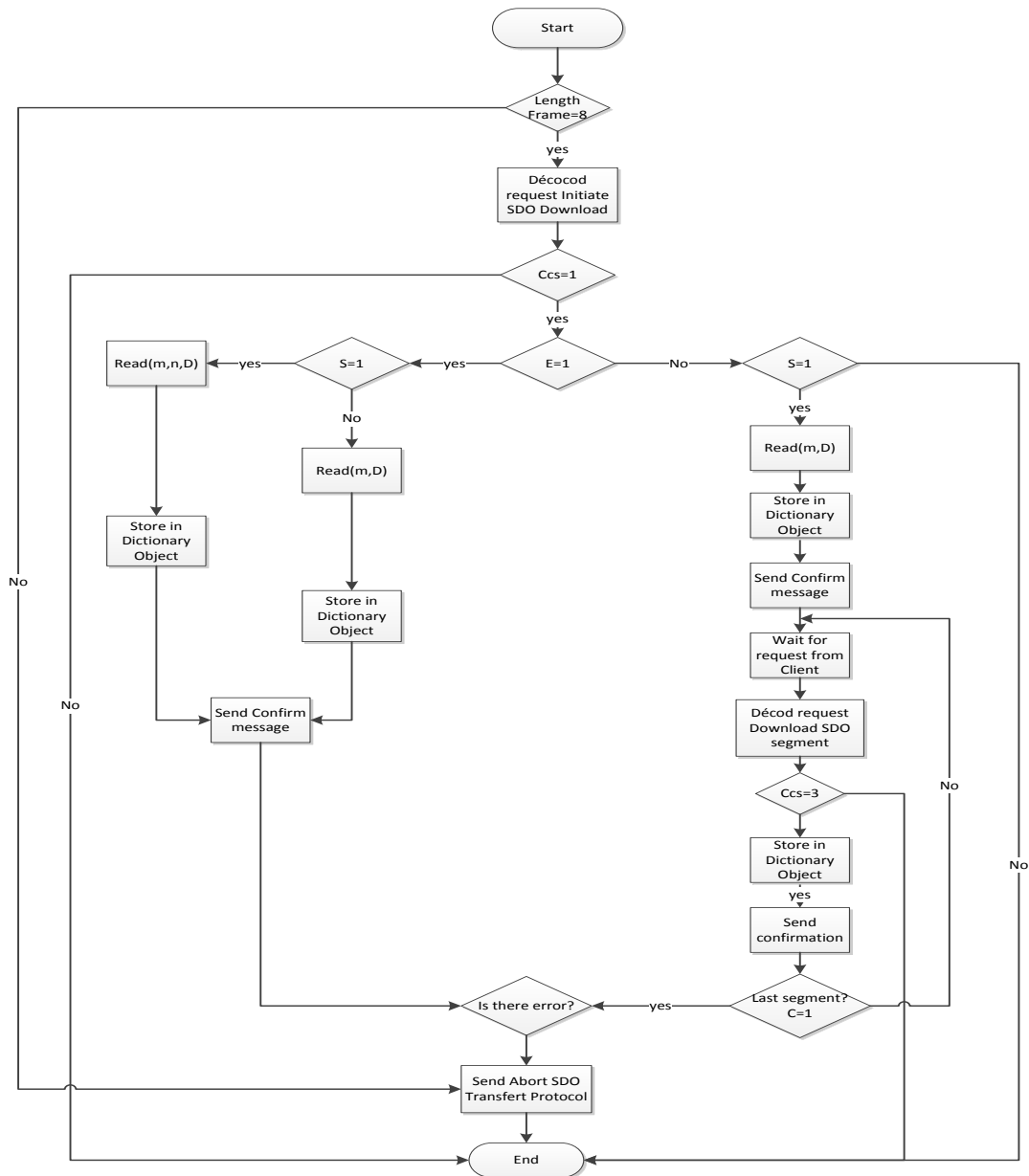


Figure III-7 : Organigramme Serveur Download SDO

III.2.8 Organigramme de la fonction Serveur upload SDO

Dans cet organigramme, le serveur fait le décodage du « Request Initiate SDO Upload » envoyé par le client. En vérifiant que le champ $ccs=2$, il lit les champs e , s et selon leurs valeurs différentes traitement sont menés :

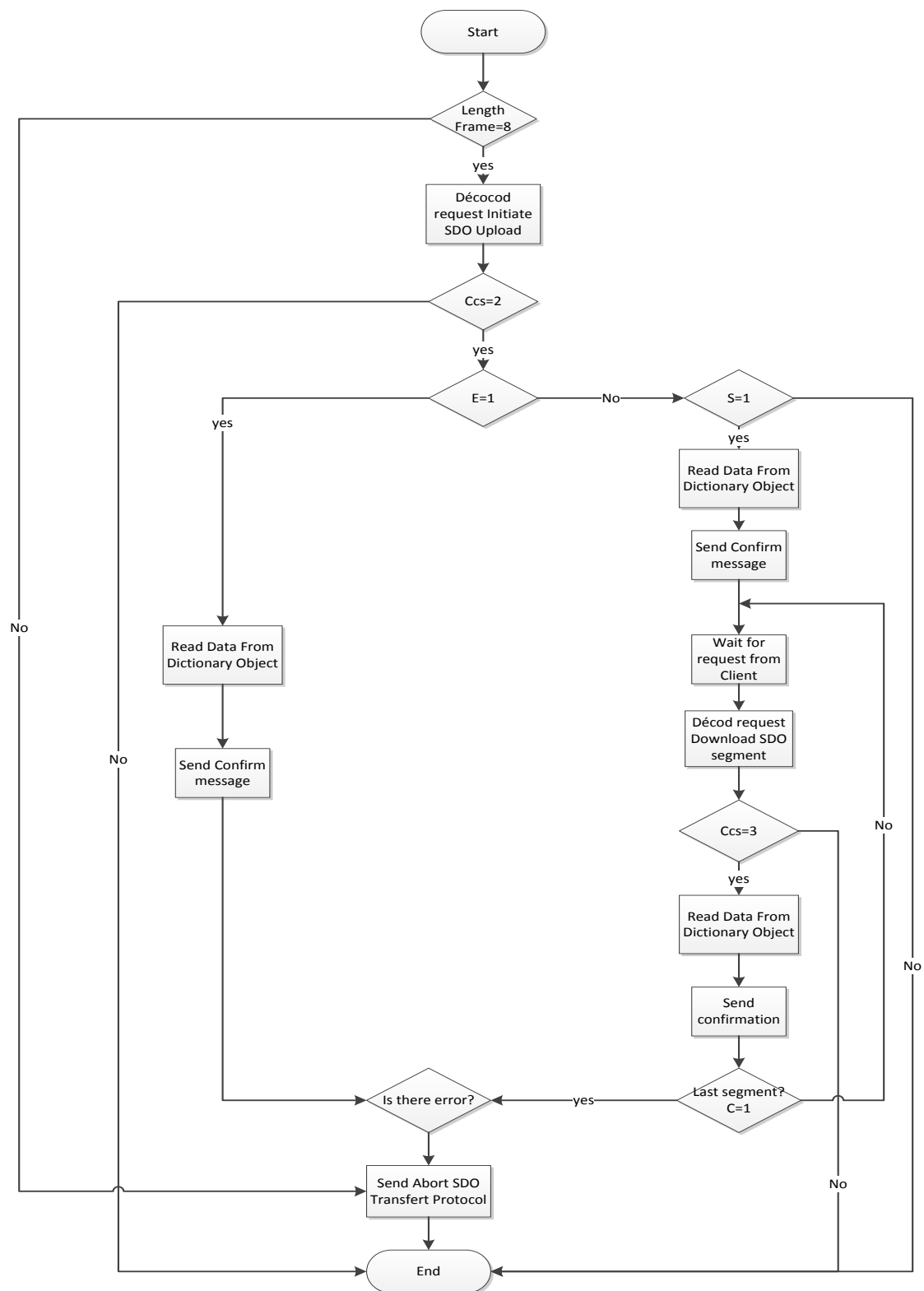


Figure III-8 : Organigramme Serveur Upload SDO

III.2.9 Organigramme de la fonction Abort SDO Transfert

Cette fonction est décrite dans la figure III-9, elle indique l'achèvement du transfert des données. Elle est utilisée pour informer le client ou le serveur des erreurs survenues lors d'un

transfert de données. La trame Abort SDO Transfert est codée sur 8 octets, le type d'erreur (Abort code) est indiqué dans le champ data de la trame.

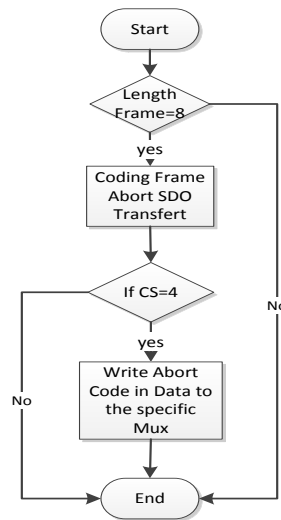


Figure III-9 : Organigramme Abort SDO

III.3 Environnements de développement hardware et software

Nous allons présenter dans ce qui suit les environnements de développement hardware et software que nous avons utilisés pour la mise en œuvre du protocole CANopen.

III.3.1 Architecture du processeur Blackfin BF548

Le processeur ADSP-BF548 d'Analog Devices est un produit de haute performance de la famille Blackfin ciblant une variété d'applications de multimédia, de l'industrie et de télécommunications [16]. Le processeur ADSP-BF548 possède jusqu'à 324Ko de mémoire sur puce. La figure III-10 montre l'architecture du processeur ADSP-BF548.

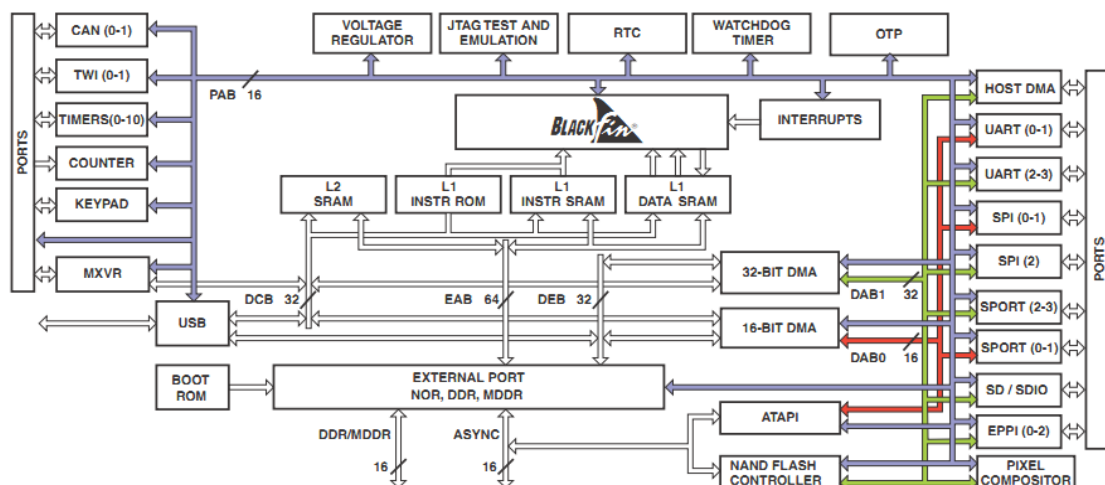


Figure III-10 : schéma synoptique fonctionnel ADSP-BF548

III.3.1.1 Architecture du cœur

La figure III.12 montre l'architecture du cœur du processeur ADSP-BF548. Le cœur Blackfin contient deux multiplicateurs/accumulateurs (MACs), deux UAL 40 bits (Unité Arithmétique et Logique), quatre UAL vidéo et un simple registre de décalage. Les unités de calcul peuvent traiter des données de 8 bits, 16 bits ou 32 bits à partir du fichier de registre.

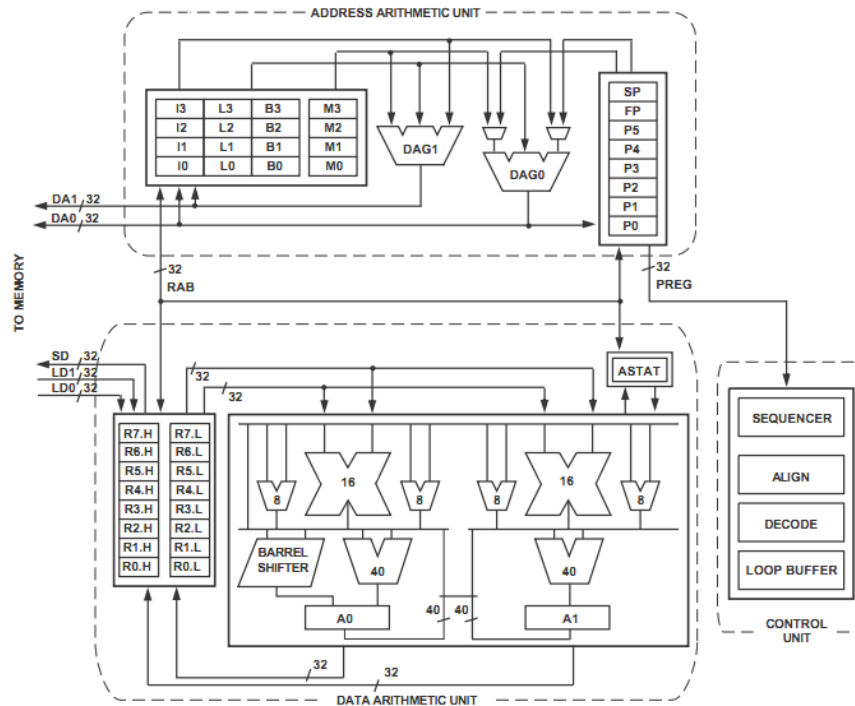


Figure III-11: Architecture du cœur ADSP-BF548

Chaque MAC effectue une multiplication 16 bits par 16 bits à chaque cycle, et le résultat est accumulé dans un registre de 40 bits. Le processeur supporte les formats signés et non signés et l'arrondissement. Les UALs effectuent des opérations classiques arithmétiques et logiques sur 16 ou 32 bits de données. De nombreuses instructions spéciales sont prévues pour accélérer certaines opérations de traitement du signal. Il s'agit notamment des opérations de bits tels que l'extraction de champ, la multiplication modulo 2^{32} , les primitives de division, le dépassement et l'arrondissement.

Le registre décalage est de 40 bits ce qui permet d'exécuter les opérations de capables de décalage, de rotation, de normalisation et d'extraction des champs.

Le programme séquenceur contrôle le flot d'exécution des instructions. Le séquenceur supporte les sauts conditionnels, les appels de sous-programmes, ainsi que les instructions ZOL (Zero-Overhead Loop).

L'unité arithmétique d'adressage fournit deux adresses pour une double récupération simultanée de la mémoire. Elle contient un fichier registre multiports composé de quatre ensembles de 32 bits associés pour les index, la modification, la longueur et les registres de base (pour mémoire tampon circulaire). Elle contient aussi huit registres de pointeurs à 32 bits supplémentaires pour la manipulation de la pile indexée en langage C.

III.3.1.2 La mémoire

Le processeur ADSP-BF548 considère la mémoire comme un espace d'adressage unifié de 4Go, utilisant des adresses de 32 bits. Toutes les ressources, y compris la mémoire interne, la mémoire externe et les registres de contrôle d'E/S, occupent des sections distinctes de cet espace d'adressage commun. La mémoire est répartie sur deux espaces : une mémoire interne et une mémoire externe. La figure III.13 montre le mappage de la carte mémoire ADSP-BF548.

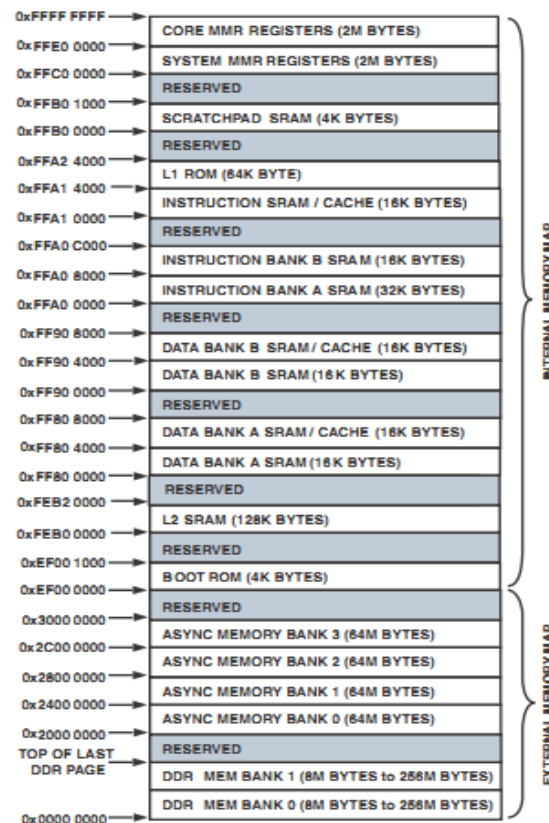


Figure III-12 : Carte mémoire ADSP-BF548

- *Mémoire interne (On-Chip) :*

Le système de mémoire sur puce L1 est la mémoire la plus performante disponible pour le processeur Blackfin. Le processeur ADSP-BF548 a plusieurs blocs de mémoire sur puce offrant un accès haut débit au cœur du processeur.

- *Mémoire externe (Off-Chip) :*

Grâce à l'unité d'interface de bus externe (EBIU), le processeur Blackfin ADSP-BF548 offre une simple connectivité pour des mémoires externes dont l'adressage est sur 16-bits. Nous citons par exemple, la DDR et DDR SDRAM mobile, SRAM, mémoire flash (type NOR, NAND) et des périphériques FIFO.

III.3.1.3 Les Périphériques

Le RTC (Real-Time Clock) est une horloge pour le processeur Blackfin ADSP-BF548, admet une fréquence externe égale à 32.768 kHz, il permet une synchronisation robuste avec les périphériques. D'autre part, le RTC peut être en tension et cadencer même si le processeur et en état de basse tension (état de veille), et ceci grâce à ces propres pins d'alimentation.

Le Processeur Blackfin ADSP-BF548 comporte une DMA (Direct Memory Access) admettant de multiples canaux indépendants. Ces canaux permettent le transfert des données automatisées avec le minimum de dépassement pour le processeur. Ce processeur contient deux unités Timer, une unité dispose de huit Timer programmables à usage général, et l'autre unité possède trois Timer.

Le processeur Blackfin intègre quatre ports séries synchrones à double chaîne (SPORT0, SPORT1, SPORT2 et SPORT3) pour les communications série et pour les communications des architectures multiprocesseurs. Aussi le processeur comporte trois ports SPI permettant la communication avec des multiples appareils SPI. D'autre part, nous pouvons trouver quatre ports (UART) full-duplex asynchrone émetteur/récepteur et deux contrôleurs CAN.

Pour notre application, nous s'intéressons essentiellement aux contrôleurs CAN [17] qui implémentent le réseau CAN 2.0B. La figure III-13 explique le schéma électrique du bus CAN.

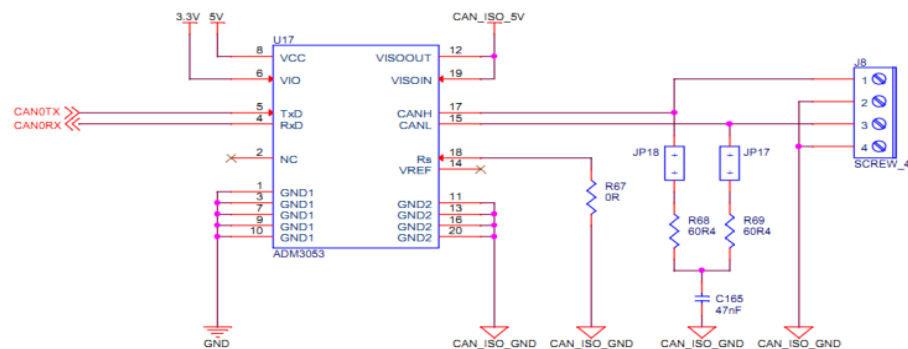


Figure III-13 : schéma électrique du bus CAN

III.3.2 Environnement de développement Visual DSP++

Le processeur Blackfin ADSP-BF548 est pris en charge avec un ensemble complet de logiciels CROSSCORE et outils de développement matériel, y compris les émulateurs d'Analog Devices et l'environnement de développement VisualDSP++. Le même matériel de l'émulateur qui prend en charge d'autres processeurs Analog Devices émule également entièrement ADSP-BF548.

L'environnement VisualDSP++ fournit les fonctionnalités suivantes :

- Édition approfondies : Créer et modifier les fichiers code source en utilisant la syntaxe du langage multiple.
- La gestion de projet : identifie les fichiers, les dépendances et les outils de développement de projet.
- l'accès à des outils de développement de code : VisualDSP++ fournit ces outils de développement de code: C / C++ compilateur, l'assembleur, l'éditeur de liens.
- Options de compilation du projet flexibles : VisualDSP++ permet de créer des fichiers ou des projets de manière sélective, mise à jour des dépendances du projet et voir l'état de votre projet en cours (signalisation des erreurs).
- Débogage : VisualDSP est un aussi un environnement de Débogage réduit le temps de débogage.

Le VisualDSP++ fournit les outils de développement suivant :

- C / C ++ compiler
- Bibliothèque avec plus de 100 en mathématiques
- Simulateur
- Emulateur



Figure III-14: VisualDSP++

III.4 Résultats d'implémentation

Dans cette section, nous allons présenter le mappage en mémoire du code CANopen que nous avons développé ainsi que les résultats de test des fonctions développées.

III.4.1 Utilisation mémoire

Pendant la phase d'implémentation du protocole CANopen, nous avons établi le mappage en mémoire du code développé. Le tableau III-1 indique le mappage de notre code sur les différentes sections de la mémoire du Blackfin ADSP-BF548. Nous remarquons que la grande partie du code et des données est placée dans la mémoire à accès rapide par le processeur notamment la mémoire L1.

Mémoire	Taille utilisée en décimale	Taille totale en décimale	Taille utilisée en pourcentage
MEM_L1_SCRATCH	76	4096	1%
MEM_L1_CODE_CACHE	16324	16384	99%
MEM_L1_CODE	3562	49152	7%
MEM_L1_DATA_B	5844	32768	17%
MEM_L1_DATA_A	32768	32768	100%
MEM_L2_SRAM	76	131072	0%
MEM_ASYNC3	76	67108864	0%

Tableau III-1 : Mémoire utilisée par le code CANopen

III.4.2 Test et validation

Le code que nous avons développé est structuré en des parties, chaque partie décrit le fonctionnement d'un objet du protocole CANopen.

Pour connaître l'émetteur d'une trame, on utilise l'identificateur des objets (COB_ID) sur le bus CAN. Le tableau III-2 décrit l'attribution des identificateurs aux objets. L'identificateur d'un nœud (Node_ID) est attribué à chaque appareil CANopen, il doit être unique dans le réseau. En général, un maître unique peut communiquer au maximum avec 127 périphériques esclaves. Alors, le maximum du NODE_ID est 7Fh qui correspond à 127 nœuds.

Objet de communication	COB_ID (Communication Object Identifier)	Nœud esclave	Exemple d'index attribué
NMT	0	Recevoir seulement	-
SYNC	128(0080h)	Recevoir seulement	1005h, 1006h, 1007h
TIME STAMP	256 (100h)	Recevoir seulement	1012h, 1013h
EMERGENCY	129+Node_ID (0081h-00FFh)	Transmettre	1014h, 1015h
T_PDO1	385+Node_ID (0181h-01FFh)	Transmettre PDO	1800h
R_PDO1	513+Node_ID (0201h-027Fh)	Recevoir PDO	1400h
T_SDO	1409+Node_ID(0581h-05FFh)	Transmettre SDO	1200h
R_SDO	1537+Node_ID(0601h-067Fh)	Recevoir SDO	1280h
Heartbeat	1792+Node_ID (0700h-077F)	Transmettre	1016h, 1017h

Tableau III-2 : les objets CANopen et leurs COB_ID attribués

Dans le tableau III-2, les objets NMT, SYNC et TIME STAMP possèdent une seule valeur d'identificateur COB_ID, ils peuvent envoyer des messages en mode diffusion. Par contre, les objets PDO, SDO et Emergency permettent l'échange de données.

Le code du protocole CANopen communique en permanence avec les couches supérieures du modèle OSI. Il y a deux fonctions qui assurent le trafic de données (Frame-Tx et Frame-Rx). La fonction Frame-Tx, permet de transmettre les paramètres suivants : l'identifiant de l'objet (COB_ID) qui est l'émetteur de la trame, le champ de données (DATA) et sa longueur correspondante (DLC). La fonction Frame-Rx, permet de recevoir les données (DATA).

Dans cette partie, nous allons tester le code du protocole CANopen avec des trames réelles pour étudier et valider sa fonctionnalité. Nous allons présenter en particulier les résultats de test des fonctions SYNC, T_SDO, T_PDO.

III.4.2.1 Résultats de test de la fonction SYNC

Pour activer la fonction SYNC, on envoie une trame Frame-Tx contenant les champs COB_ID, Data et DLC. Le champ COB_ID est égal à 80h et le champ Data est vide d'où la longueur des données DLC=0. On charge les adresses des objets de la fonction SYNC du Dictionnaire Objet dans un tableau appelé *SYNC*, ensuite on cherche l'adresse d'Index 1006h, qui correspond à la période d'envoi de la trame de Synchronisation.

La trame de synchronisation SYNC est broadcastée périodiquement chaque *Sync_counter* (période d'envoi). Pour réaliser cette période, nous avons créé une fonction *Timer()* permettant de faire l'incrémentation temporelle d'une milliseconde jusqu'à atteindre la valeur du *Sync_counter*. Dans la figure III-15, un exemple de période d'envoi du SYNC égale à 0Ah est considéré.

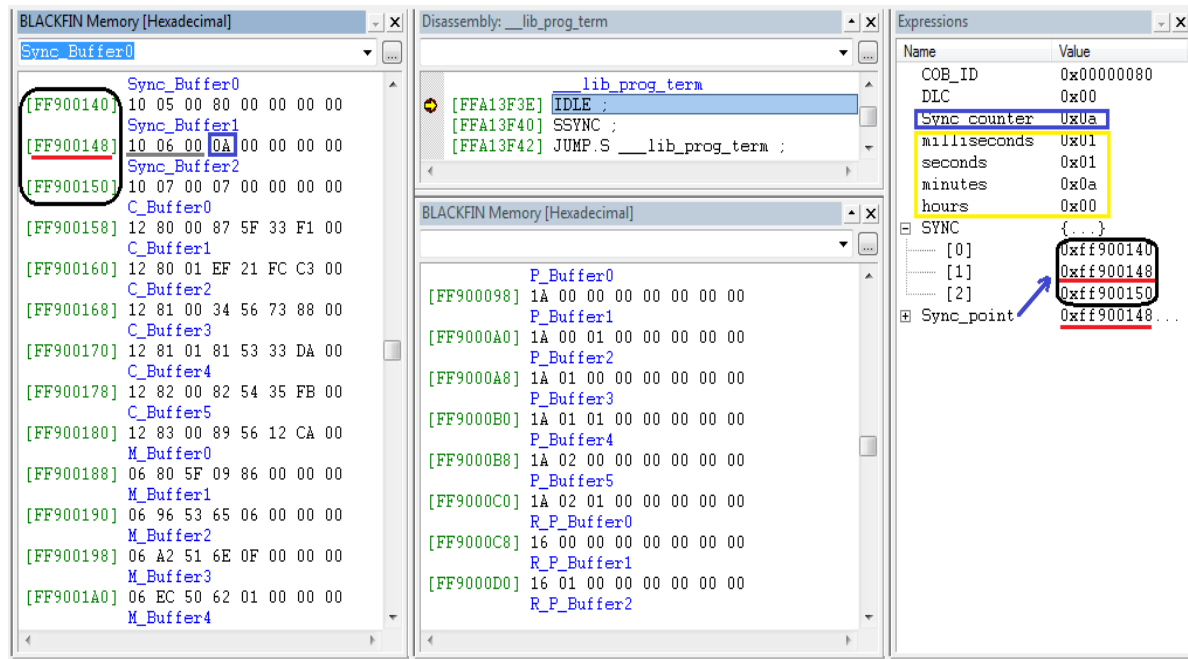


Figure III-15 : Test de la fonction SYNC

III.4.2.2 Résultats de test de la fonction *T_PDO*

Dans le traitement transmettre PDO, on envoie une trame Frame-Tx (COB_ID, Data, DLC). Le champ COB_ID doit être dans l'intervalle [0x0180, 0x01FF], le champ Data peut contenir des données jusqu'à 8 octets et le champ DLC ne doit pas dépasser 08h.

Pour faire le mappage des données, nous avons passé par les étapes suivantes. On charge dans le tableau *T_pdo* les adresses des objets enregistrés dans le Dictionnaire Objet, ensuite, on accède aux adresses qui vont être mappées et on enregistre leurs Index et Sub-Index dans des variables Mux. Nous avons créé une fonction *Mapping_appel()* permettant de mapper les données des Mux dans une seule variable appelée *mapping*. Enfin, on stocke la donnée mappée dans le Dictionnaire Objet à l'index 1A00h.

La trame PDO est générée si on reçoit un événement SYNC, ou on reçoit une trame Remote Frame de R_PDO. Le BufferSend prend la valeur de la donnée mappée et l'envoi dans le champ Data du Frame-Tx.

Dans la figure III-16, un exemple de mappage des données de deux Mux est présenté. Il s'agit des données 0986h pour Mux 1 et 6201h pour Mux2.

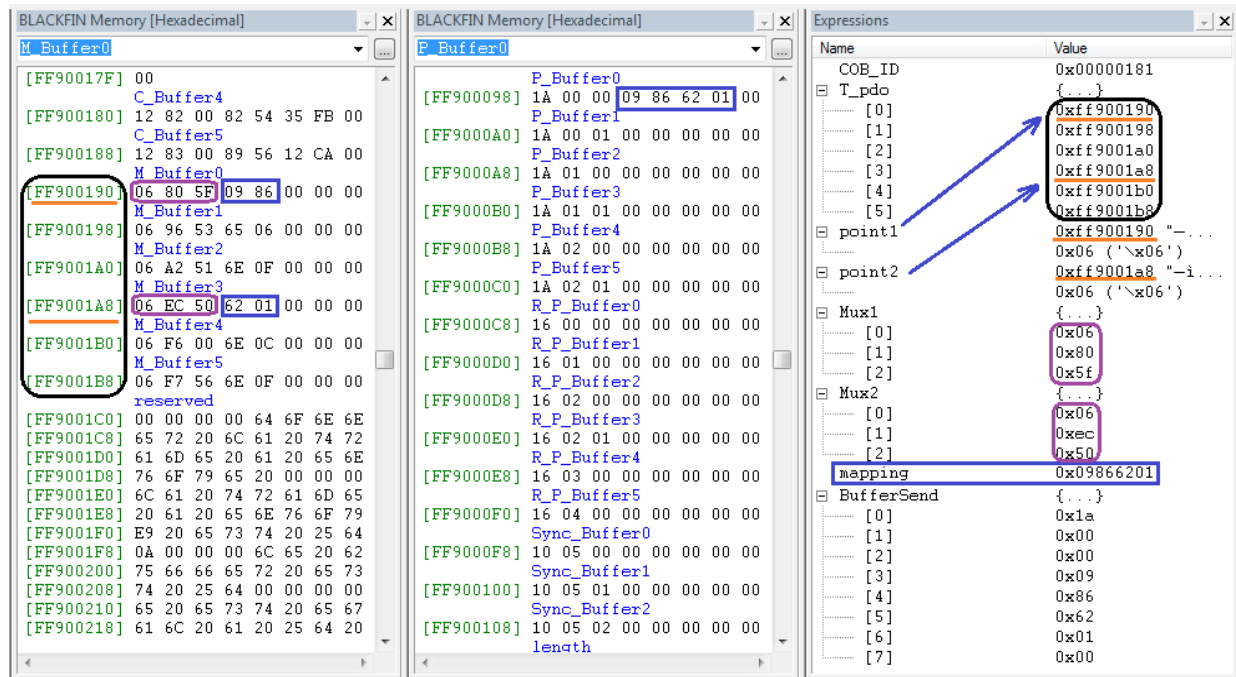


Figure III-16 : Test de la fonction Transmettre PDO

III.4.2.3 Résultats de test de la fonction T_SDO

Dans le traitement transmettre SDO, on envoie une trame Frame-Tx (COB_ID, Data, DLC). Le champ COB_ID doit être dans l'intervalle [0x0580, 0x05FF], le champ Data doit contenir 8 octets de données et le champ DLC contient la valeur 08h.

Les trois premiers bits du premier octet de la donnée Data correspondent à la variable ccs permettant de choisir soit Download SDO ou Upload SDO. Dans ce qui suit, nous allons présenter les résultats de test de la fonction Download SDO pour deux cas de figure d'envoi de données accéléré et normal définis par la variable e. Il s'agit des fonctions Initiate SDO Download et Download SDO Segmented.

III.4.2.3.1 Fonction Initiate SDO Download

Pour faire l'envoi d'une trame Initiate SDO Download, le client prépare la structure CID_SDO (Client Initiate Download SDO), il met les champs ccs à 1, e à 1 (pour l'envoi accéléré), n à 0 (n indique le nombre d'octets qui ne contiennent pas de données) et s à 1 (pour indiquer que la taille de données est spécifiée). La valeur totale du premier octet CS (Command Specifier) est égale à 23h. Par la suite, on charge dans le tableau C_sdo les adresses des objets enregistrés dans le Dictionnaire Objet, ensuite on accède à l'adresse qui va

être envoyée et on enregistre ses Index et Sub-Index dans une variable Mux (3 octets) ainsi que ses données dans le champ Data (4 octets). Enfin, on remplit les 8 octets dans le *BufferSend* pour l'envoyer au serveur.

Le serveur répond le client par une trame de confirmation dans le *Buffer*. Le client décode ce message dans la structure *SID_SDO* (Server Initiate Download SDO), il vérifie que le champ *scs*=3 et que le champ Mux est le même que le champ envoyé. La figure III-17 montre un exemple de test de la fonction Initiate Download SDO.

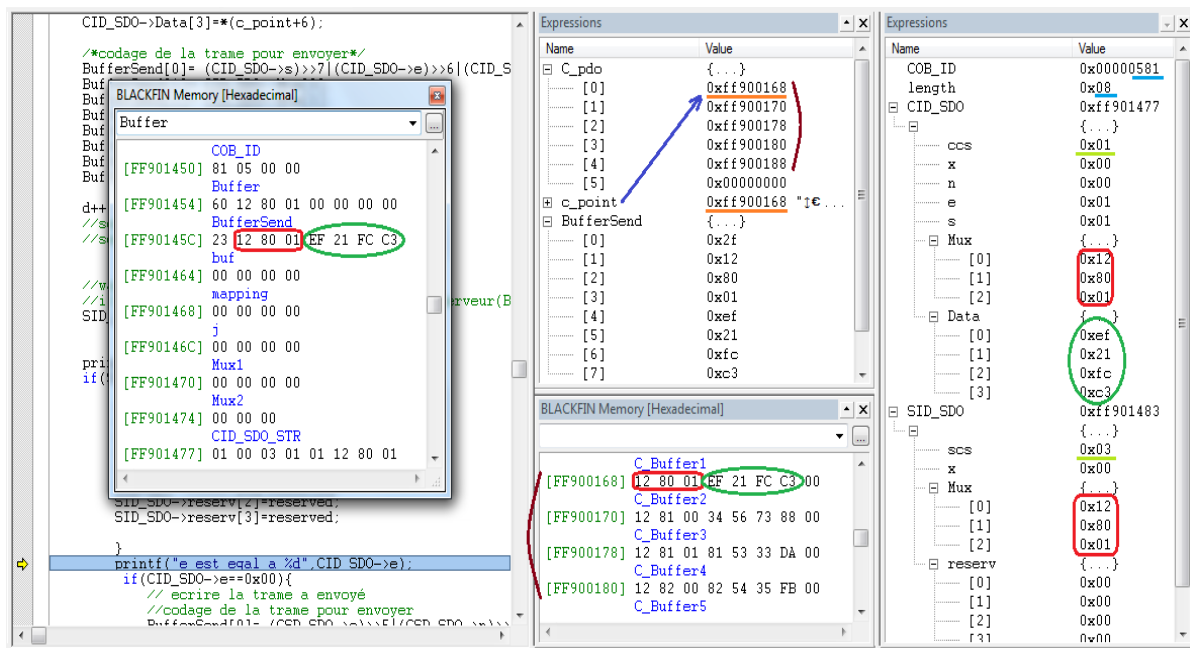


Figure III-17 : Test de la fonction Initiate Download SDO

III.4.2.3.2 Fonction Download SDO Segmented

Pour faire l'envoi d'une trame Download SDO Segmented, on fait le même traitement que celui effectué pour l'envoi d'une trame Initiate Download SDO. On change la variable *e* à 0 pour indiquer qu'il s'agit d'un transfert segmenté et on envoie seulement dans le champ Data la taille de l'objet à envoyer.

Le client prépare la structure *CSD_SDO* (Client Segmented Download SDO) pour envoyer les données, il met le champ *ccs* à 0 (indique qu'il s'agit d'une trame de transfert de données segmentées), *t* initialement à 0 (puis bascule à chaque envoi d'une nouvelle trame), *n* à 3 (indique le nombre d'octets qui ne contiennent pas de données) et *c* à 1 (indique qu'il n'y a pas d'autres segments à envoyer). La valeur totale du premier octet est égale à 07h. On remplit par la suite les valeurs de *seg_data* sur 7 octets et à la fin on envoie au serveur les 8 octets rassemblés dans le *BufferSend*.

Le serveur répond le client par une trame de confirmation dans le *Buffer*. Le client décode ce message dans la structure *SSD_SDO* (Server Segmented Download SDO), il vérifie que le champ *scs*=1. La figure III-18 montre un exemple de test de la fonction Download SDO Segmented.

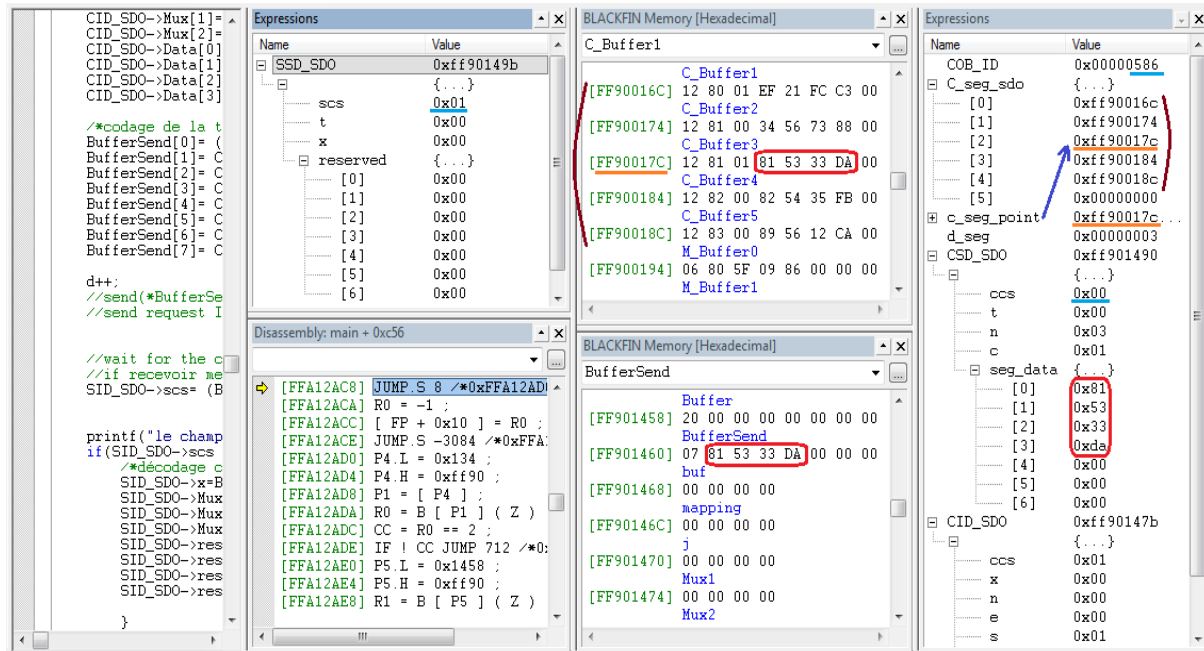


Figure III-18 : Test de la fonction Download SDO Segmented

III.5 Conclusion

Dans ce dernier chapitre, nous avons présenté les différents organigrammes détaillant le fonctionnement du CANOpen. Ensuite, nous avons présenté les environnements de développement hardware et software de notre application. Enfin nous avons présenté les résultats de tests de l'implémentation du code CANOpen. Nous avons présenté en particulier les résultats de validation des fonctions SYNC, T_PDO et T_SDO pour les modes d'envoi normal et accéléré. La grande partie du code que nous avons développé est placée dans la mémoire à accès rapide par le processeur notamment la mémoire L1.

Conclusion générale

Le présent rapport est rédigé dans le cadre de la préparation d'un projet de fin d'études à l'Institut Supérieur d'Informatique, en vue de l'obtention de diplôme de licence appliquée en informatique industrielle, spécialité Systèmes Embarqués.

Notre projet consiste à la mise en œuvre et l'implémentation du protocole CANopen sur une plateforme DSP du type Blackfin BF548 pour la société EBSYS. Nous avons implémenté en particulier les interfaces de communication CANopen pour la gestion de réseau (NMT, SYNC, Emergency, Time Stamp), le contrôle de l'état des nœuds (Heartbeat) et l'accès au dictionnaire objet (PDO, SDO). Les tests du code que nous avons développé ont été concluants et ont permis de valider l'implémentation du protocole sur le DSP.

EBSYS dispose déjà des couches qui gèrent la l'envoi et la réception des trames sur le bus CAN. Comme perspectives de notre travail, nous envisageons d'intégrer sous l'environnement UNIX le code que nous avons développé avec le code qui gère la communication sur le bus CAN et de tester son fonctionnement sur la carte ezLINX. Enfin, nous essayerons de faire une optimisation du code en langage assembleur.

Ce projet nous a été bénéfique sur le plan professionnel et personnel. Sur le plan professionnel, nous avons eu l'occasion d'aiguiser nos connaissances en langage C et en plateforme de développement DSP et de travailler sur un projet de grande envergure.

Sur le plan personnel, nous avons appris à bien travailler en équipe, à se faire des concessions et de se familiariser avec l'environnement professionnel.

Bibliographie

- [1] <http://www.ebsys-tech.com/>
- [2] <http://www.oberle.org/can-can.html>
- [3] <http://www.maxonjapan.co.jp/manual/epos/EPOS-CommunicationGuide-E.pdf>
- [4] http://pic18.free.fr/contenu2download/bus_can/can.ppt
- [5] <http://www.technologuepro.com/cours-systemes-embarques/cours-systemes-embarques-Bus-CAN.htm>
- [6] http://lmi92.cnam.fr:8080/NSY209/projets/projet10_CAN_Bluetooth/bibliographie/CAN_TechniquesIngenieur.pdf
- [7] http://www-igm.univ-mlv.fr/~dr/XPOSE2009/BusCAN/intro_can.html#p3
- [8] http://moodle2.diderot.org/pluginfile.php/3846/mod_resource/content/0/1-busCAN-trame.ppt
- [9] Dominique, Paret « *Réseaux multiplexés pour systèmes embarqués : CAN, LIN, FlexRay, Safe-by-Wire...* », Dunod, 2012.
- [10] <http://www.canopensolutions.com/index.html>
- [11] <http://www.a2v.fr/program/canopen.htm>
- [12] http://www.ixxat.com/can_application_layer_introduction_en.html
- [13] http://reperes-formation-mei.lyceehfontaine.fr/collections/1/ligne%20RECYCLICC%20SIMPLEX/Programme%20automate/simplex/586-fr-NU-eNod1-3T-CAN-F-1010_165787-A.pdf
- [14] http://www.kuebler.com/PDFs/Feldbus_Multiturn/CANopen/busprofil_CAN.pdf
- [15] http://www.systemec-electronic.com/uploads/87/c8/87c8ebed461e58d6509aaec9b87f1c28/L-1020e_12_truncated.pdf
- [16] http://www.analog.com/static/imported-files/data_sheets/ADSP-BF542_BF544_BF547_BF548_BF549.pdf
- [17] http://www.analog.com/static/imported-files/eval_boards/ezLINX_Schematic.pdf