C3-3 DEFINIR L'ARCHITECTURE GLOBALE D'UN PROTOTYPE OU D'UN SYSTEME.
C3-4 VALIDER LE CHOIX D'UNE ARCHITECTURE MATERIELLE/LOGICIELLE.
C4-1 CABLER ET/OU INTEGRER UN MATERIEL.
C4-3 ADAPTER ET/OU CONFIGURER UNE STRUCTURE LOGICIELLE.
C4-5 TESTER ET VALIDER UN MODULE LOGICIEL ET MATERIEL.

Système anticollision automobile

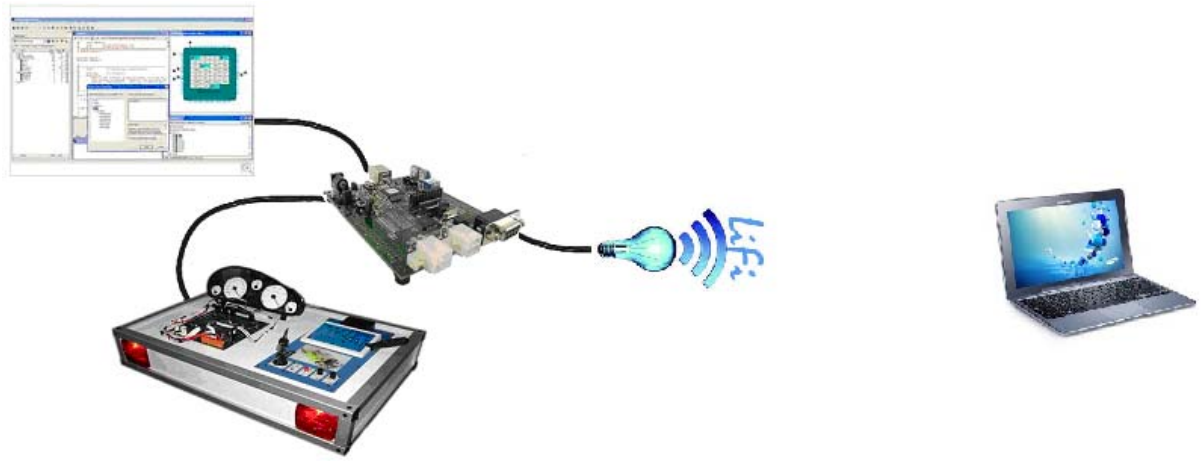
1 Présentation :

L'augmentation de la circulation ainsi que l'inattention de certains conducteurs aggrave un problème de sécurité. L'électronique embarquée dans les véhicules permet de remédier à certaines défaillances des conducteurs afin d'éviter une collision avec le véhicule précedent.

2 Objectif :

Le but de notre travail est de faire communiquer un véhicule avec celui qui le suit afin d'échanger des paramètres importants pour la sécurité. Nous nous intéresserons à la transmission de la vitesse. La transmission se fera en LIFI.

Afin de simplifier et de limiter le travail, le véhicule de devant sera simulé par un module Exxotest, le véhicule de derrière (récupérant les données de vitesse) sera un PC équipé du logiciel COMM (ou hyperterminal).



Pour obtenir ce résultat, un technicien doit réaliser les taches techniques suivantes :

Gestion de l'émetteur :

Connecter une carte à µcontrôleur sur un bus CAN du véhicule (de devant)
Configurer le bean « FreescaleCAN » du µcontrôleur.
D'implémenter le code de communication afin de lire les trames et d'en extraire les données.
Connecteur un module émetteur LIFI sur la carte à µcontroleur
Configurer le bean « Term » (liaison série)
Implémenter le code de communication de la liaison série.

Gestion du récepteur :

Connecter le module Récepteur LIFI sur le PC
Recevoir les informations de vitesse sur COMM

Nous vous proposons dans ce TP de réaliser l'ensemble des opérations vous permettant d'extraire les données contenues dans les trames CAN en vue de leur récupération sur COMM.

1ère Etape : Analyse des connexions entre les divers modules

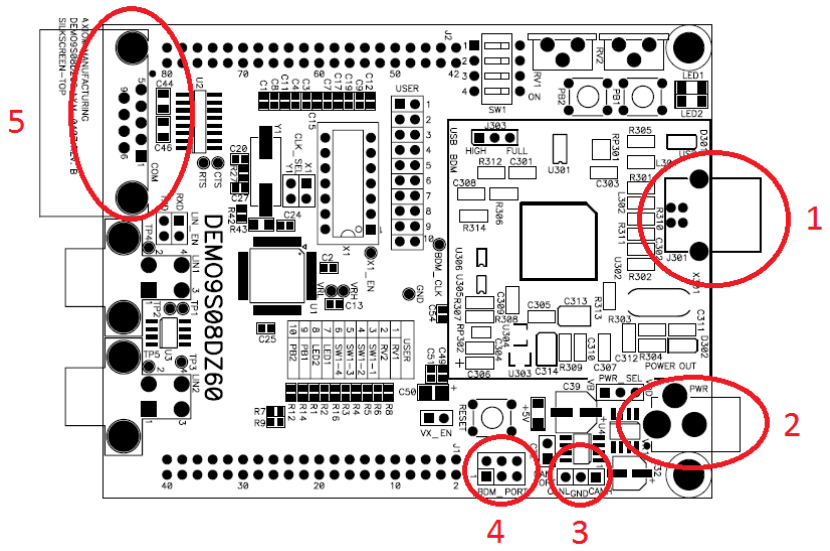
Compléter le diagramme de bloc interne (présent à la fin du document).

2ème Etape : Connecter une carte à µcontrôleur sur un bus CAN

A partir de la documentation de la carte Demo9S08DZ60, identifier sur le schéma ci-dessous les numéros correspondant aux connecteurs d'E/S suivants:

- Alimentation
- Entrée de programmation (vous différentierez l'USB et le BDM)
- Port de communication série
- Port de communication CAN (vous différencierez le CANH et le CANL).

N°	connecteur
1	
2	
3	
4	
5	



Connecter la carte Demo9S08DZ60 à l'ordinateur (qui permettra la programmation) ainsi qu'au bus CAN.

3ème Etape : Configurer le bean « FreescaleCAN » du µcontrôleur

On utilisera CodeWarrior 10.6 afin de programmer la carte Demo9S08DZ60.

Créez un projet « LIFI », vous choisirez le bon µcontrôleur (correspondant à celui de la carte).

Insérer un bean « Freescale CAN » et le configurer conformément à la fiche de guidance « Configuration bean Freescale CAN ».

4ème Etape : Implémenter le code de communication afin de lire les trames et d'en extraire les données.

a - Implémenter le code de communication afin de lire les trames

La lecture des données circulant sur le bus se fera par interruption. A partir de la fiche de guidance, identifier l'évènement et la méthode à utiliser (deux).

Evènement :
Méthode :

Créez les variables suivantes dans le fichier main.c et **intégrez des commentaires pour définir leur rôle** :

```
/* User includes (#include below this line is not maintained by Processor Expert)
*/
              char ID[4];
              char Frametype[1];
              char FrameFormat[1];
              char length[1];
              char data[8];
```

Intégrer le code suivant dans l'évènement « CAN1_OnFullRxBuffer » **et intégrer des commentaires.**

```
void CAN1_OnFullRxBuffer(void)
{
              extern char ID[4];
              extern char Frametype[1];
              extern char FrameFormat[1];
              extern char length[1];
              extern char data[8];

              CAN1_ReadFrame(ID,Frametype,FrameFormat,length,data);
}
```

Après compilation, mettre un point d'arrêt après l'instruction « `CAN1_ReadFrame` ».
Relancer plusieurs fois l'exécution du programme jusqu'au point d'arrêt.
Relevez la valeur de l'identifiant reçu.
Conclure sur ces différentes valeurs.

b - Implémenter le code de communication afin d'extraire les données.

Principe d'extraction :

Lors de la réception d'une trame sur le bus CAN, celle-ci est stockée selon son identifiant dans un (ou plusieurs) tableau de données. Ce tableau est donc mis à jour en continu lors de la réception d'une nouvelle trame.

Les variables contenues dans les trames sont ensuite reconstruites à partir du tableau de données.

Ex : Lors de la réception de la trame 3E1 sur le bus CAN, les 6 octets contenus dans la trame reçue vont être stockés dans un tableau trame3E1[8] (tableau contenant 8 caractères). Ensuite, les variables regime, vitesse,boite et état_mot doivent être mise à jour automatiquement grâce aux données contenues dans le tableau trame3E1.

Compéter les pointillés ci-dessous afin d'extraire la valeur de régime à partie des valeurs contenus dans le tableau trame3E1

regime = trame3E1[…..] *…… + trame3E1[……];

Modifier le code dans la partie de réception de trame par interruption afin d'obtenir le code suivant:

```
void CAN1_OnFullRxBuffer(void)
{
              extern char ID[4];
              extern char Frametype[1];
              extern char FrameFormat[1];
              extern char length[1];
              extern char data[8];
              extern long Ident;

              extern char trame3E1[8];
              extern int regime;
              extern int vitesse;
              extern char boite;
              extern char etat_mot;

              char i;

                    // Lecture de la trame reçue
                    CAN1_ReadFrame(ID,Frametype,FrameFormat,length,data);


                    // récupération de l'identifiant sous forme d'entier
                    // celui-ci est récupéré sur 32 bits par défaut par la fonction
                    // CAN1_ReadFrame afin d'être compatible quel que soit le
                    // mode choisi (mode standard ou mode étendu)
                    Ident =ID[0]*16777216 +ID[1]*65536 + ID[2]*256 +ID[3];

                    // Stockage de la trame reçue,
                    if (Ident == 0x3e1)
                                {
                                for (i=0;i<8;i++) trame3E1[i] = data[i];
                                }
                    // extraction des données
regime =      ..... à compléter..... ;
vitesse =     ..... à compléter..... ;
boite =       ..... à compléter..... ;
etat_mot =    ..... à compléter..... ;
}
```

Les parties « à compléter » seront bien évidemment complétées.
Les variables manquantes seront bien sur rajoutées dans le main.c.

Remarque : le descriptif des trames utilisées par le module Exxotest sont données dans le fichier « Notice trames entre les platines.pdf »

Enlever les points d'arrêts et lancer le programme en continu.
Faire varier la vitesse du module moteur, et valider le fonctionnement du système (mise à jour automatique des variables « regime, vitesse,boite, etat_moteur ».

5ème Etape : Connecter un module émetteur LIFI sur la carte à µcontroleur.

Réaliser le câblage (entre l'émetteur et la carte DZ60) et alimenter l'émetteur LIFI

6ème Etape : Configurer le bean « Term » (liaison série)

Dans le projet code warrior précédent, insérer un bean « Term » et le configurer conformément à la fiche de guidance « Configuration bean Term »

Remarque : la vitesse de transfert doit être réglée sur 115200 bauds.

7ème Etape : Implémenter le code de communication de la liaison série.

On souhaite envoyer la valeur de la variable « vitesse » sur la liaison série (sur laquelle est branchée le module LIFI).

Citer la méthode (du module Term) à utiliser pour envoyer directement un nombre sur la liaison série:

Rajouter donc les lignes suivantes dans le code

```
void CAN1_OnFullRxBuffer(void)
{
            extern char ID[4];
            extern char Frametype[1];
.
.
.
regime =     ..... à compléter..... ;
vitesse =    ..... à compléter..... ;
boite =      ..... à compléter..... ;
etat_mot =   ..... à compléter..... ;


Term1_SendNum(vitesse);
Term1_CRLF();
}
```

Lignes à rajouter

8ème Etape : Connecter le module Récepteur LIFI sur le PC

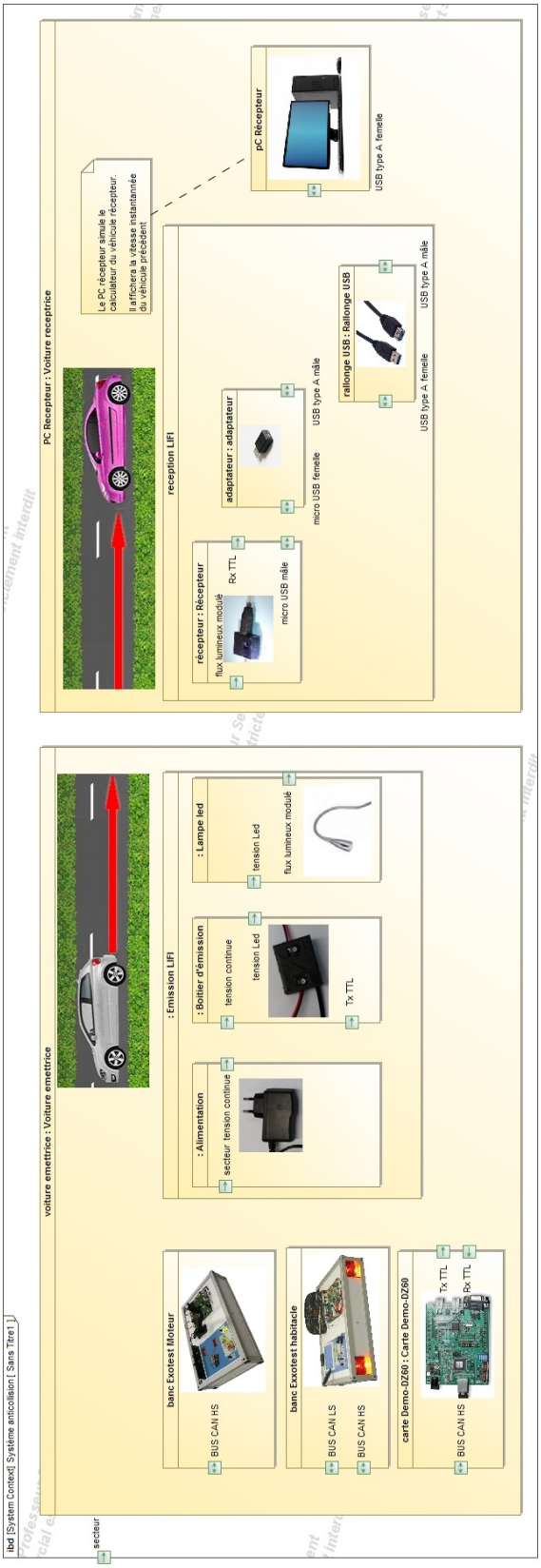
Réaliser le câblage (entre le récepteur LIFI et le PC de réception).

Positionner le module récepteur en face du module émetteur et à distance convenable.

9ème Etape : Recevoir les informations de vitesse sur COMM

Démarrer le programme COMM, le paramétrer sur le port ou est branché le module LIFI, paramétrer la vitesse de transmission à 115200 bauds.

Valider la valeur reçue en fonction de la vitesse du véhicule précédent (module Exxotest).


ibd [System Context] Système anticollision [Sans Titre]
PC Recepteur : Voiture receptrice
pC Récepteur
USB type A femelle
Le PC récepteur simule le calculateur du véhicule récepteur. Il affichera la vitesse instantannée du véhicule précédent
reception LIFI
récepteur : Récepteur
flux lumineux modulé
Rx TTL
micro USB mâle
adaptateur : adaptateur
micro USB femelle
USB type A mâle
rallonge USB : Rallonge USB
USB type A femelle
USB type A mâle
voiture emettrice : Voiture emettrice
: Emission LiFi
: Lampe led
tension Led
flux lumineux modulé
: Boitier d'émission
tension continue
tension Led
Tx TTL
: Alimentation
secteur tension continue
banc Exotest Moteur
BUS CAN HS
banc Exxotest habitacle
BUS CAN LS
BUS CAN HS
carte Demo-DZ60 : Carte Demo-DZ60
Tx TTL
Rx TTL
BUS CAN HS
secteur