TP Informatique Industrielle
Utilisation du protocole CAN

Ce TP va permettre d'expérimenter des échanges de données via le protocole CAN (Controller Area Network). Ce principe d'échange est très utilisé dans le milieu industriel en particuliers dans l'automobile.

La carte µContrôleur utilise un CPU 80C592. Ce dernier possède un contrôleur CAN intégré. Pour utiliser le contrôleur, on dispose de plusieurs registres. Ces registres vont nous fournir des informations sur l'état du contrôleur CAN (par exemple des erreurs éventuelles, de la réception d'un message, …). Ils permettent aussi de configurer ce contrôleur.

Pour faciliter le déroulement de ce TP et compte tenu du temps imparti, nous avons réalisé une couche logicielle CAN (noté CAN-NET). C'est un ensemble de fonctions de base permettant d'initialiser le CAN, de réaliser un envoi et une réception de trame dans un buffer (un tableau de char). Un ensemble de paramètres (#define déclarés dans le fichier " **candefs.h** ") permet de configurer simplement les constantes essentielles (identifiant du module, Masque de filtrage d'adresses ….etc). Cette couche logicielle, désignée par HAL (Hardware Abstraction Layer) met en œuvre les interruptions pour gérer le contrôleur CAN et récupérer une trame utile dès l'arrivé de celle ci dans le contrôleur. En résumé, la couche logicielle " CAN-NET " prend en charge les actions suivantes :

- l'utilisation du DMA ;
- l'utilisation du bus en mode différentiel ;
- l'adresse du contrôleur ainsi que le registre de masquage (à condition de les configurer dans le fichier include);
- la vitesse de transmission ;
- l'utilisation du contrôleur par interruption, …

Il est important de noter que les événements du contrôleur CAN sont reproduits en positionnant des FLAGs déclarés de type bit. Ces FLAGS, lorsqu'ils sont positionnés, indiquent que tel action s'est produite par le contrôleur CAN. A titre d'exemple :

- le bit **can_rx_ready** permet d'indiquer qu'une nouvelle trame CAN est disponible dans le tableau de données **can_buffrx []**.
- Lorsque le bit **can_tx_ready** est positionné, cela signifie qu'une émission est possible. Nous utiliserons pour cela la fonction **send_message(….)** en passant le tableau de la trame en paramètre.
- Lorsque le bit **can_init_request** est positionné à TRUE, cela permet d'initialiser le contrôleur CAN. La configuration du contrôleur CAN est effectuée dans la fonction *can_config ( )*.

Le programme principal ci-dessous, donne un exemple de programme minimal permettant d'initialiser correctement le contrôleur CAN et de se prémunir d'erreurs possibles. Dans ce dernier cas, le contrôleur est réinitialisé.

```
void  main()
{
        CANInitialisation();                    // initialisation du contrôleur CAN
        while(1)
        {
                if ( can_system_error ){        // erreur donc demande d'initialisation
                        can_init_request = TRUE;// demande d'initialisation
                }
                if ( can_init_request ){        // besoin d'une initialisation
                        CANInitialisation();
                        can_init_request=FALSE;
                }
        }
}
```


Le TP va se dérouler en deux étapes. La première consistera à s'intéresser à la partie réception. La seconde permettra de se familiariser avec l'émission en répondant à des requêtes.

Un module principal CAN est relié au réseau. Ce module va envoyer périodiquement un message avec un identifiant prédéfini.

1) Nous souhaitons récupérer les données envoyées et les afficher sur la console du Débuggeur. Pour cela, on vous fournit le code minimum dans le programme principal. L'interruption du contrôleur CAN se déclenche dans 3 cas : lors de la réception d'un message, lorsqu'une transmission est réalisable (buffer d'émission disponible) et lorsqu'il y a une erreur. Dans le cas d'une réception d'une trame, un flag logiciel can_rx_ready (mis en œuvre dans la couche logicielle qui vous est donnée) est positionné à 1. Votre programme devra alors récupérer la trame CAN et l'afficher (en hexa) sur la console texte de µvision.

   Ecrire le code du programme principal qui permet de lire les données. Attention, il ne faudra pas oublier de mettre les bonnes valeurs dans les registres d'adresse et de masquage.

2) Dans cette partie, le module CAN principal va émettre des requêtes. Il faudra alors répondre à cette requête en renvoyant un message précis. Pour cela, vous devez placer dans un tableau les données à émettre puis appeler la fonction *unsigned char send_message (unsigned char *tabtx)*. Des explications seront données en TP.

3) A partir de ce qui a été fait, écrire un programme permettant de converser avec une autre station CAN. Nous utiliserons alors les principes suivants :
   - Chaque station CAN dispose d'un identifiant propre ;
   - La taille du message envoyé à une station sera inférieure à 30 octets. Compte tenu de la taille d'une trame CAN élémentaire (8 octets). Un fractionnement du message en trames sera nécessaire. Il faudra proposer un moyen de contrôle pour s'assurer que toutes les trames sont reçues.
   - Envoyer un message texte (par exemple) à plusieurs nœuds et de façon périodique.

4) Emuler un allumage automatique progressif des feux d'un véhicules en utilisant un capteur de lumière.