UTILISATION DU BUS CAN POUR UN SYSTÈME TEMPS RÉEL DE COMMANDE DISTRIBUÉE

1) BUT DE LA MANIPULATION

Ce TP a pour objectif final d'être capable de mettre distribuer une application logicielle sur 3 abonnés à un réseau de type CAN en se basant exclusivement sur les services de communication de la couche liaison de données du protocole CAN.

- Partie 1 : Prise en main du contrôleur CAN du C167 par l'implémentation de fonctions de bases de communication pour un utilisateur, en employant des trames de données et des trames des requêtes.
- Partie 2 : implémentation d'un système de contrôle basique en se basant sur les services de la couche liaison

2) PRÉSENTATION GÉNÉRALE DU PROBLÈME

L'application à distribuer est un système de contrôle très simple avec E/S déportées au travers d'un réseau CAN. Le cycle de l'algorithme de contrôle se limite à la lecture de la valeur du capteur, suivi du calcul de la commande et de l'envoi de la commande à l'actionneur (voir figure 1-a)). Dit autrement, un **calculateur (abonné 1)** doit utiliser la mesure **d'un capteur (abonné 2)**, réaliser un calcul partir de cette mesure, pour déterminer la valeur de la commande à appliquer un **actionneur (abonné 3)**.

Figure 1 – a) Système de commande à mettre en œuvre - b) – Transposition « en bus CAN »

Le réseau sur lequel est bâti le système de contrôle à mettre en œuvre est constitué de 3 stations C167CR reliées à un bus CAN (Figure 1-b) :

- le contrôle (calcul de la commande) tourne sur la station S1
- un capteur est relié à la station S2,
- un actionneur est relié à la station S3

3) PRÉSENTATION DE L'APPLICATION UTILISÉ

La figure 1-a) présente une vue schématique du système (maquette) utilisée comme support de l'application à distribuer. Cette maquette simule un système simple de commande de moteur de papillon par l'intermédiaire d'un capteur de position (potentiomètre) de la pédale d'accélérateur.

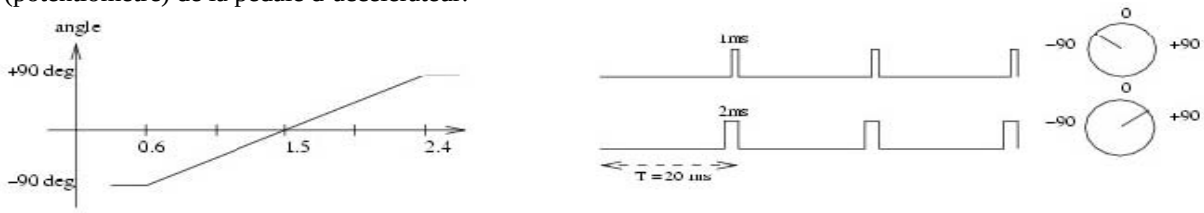


Figure 2 – a) Correspondance largeur d'impulsion – angle b) Signaux et angles associés

Un **potentiomètre** est considéré comme étant associé à la pédale d'un accélérateur retourne une grandeur analogique proportionnelle à la position de la pédale. Le **moteur du papillon** est un servo-moteur à commande PWM. Un servo-moteur désigne un dispositif utilisé en modélisme constitué d'un petit moteur à courant continu et d'un asservissement de position électronique. Le signal de commande qui pilote le moteur est une consigne de position, le moteur tourne d'un angle proportionnel à la consigne et s'arrête lorsque la position cible est atteinte. Ce signal de commande est un signal [0;5]V module en largeur de période 20ms (PWM). Le servo-moteur prend en compte la largeur temporelle de l'impulsion qu'il convertit proportionnellement en un angle. La position 0° correspond une largeur d'impulsion de 1,5ms; par rapport à cette largeur. Une variation de +0.9ms (largeur 2,4 ms) correspond à un angle de +90°, une variation de -0.9ms (largeur 0,6 ms) correspond à un angle de -90°. Les figures ci-dessus illustrent la correspondance entre largeur d'impulsion et angle. Le signal est dit modulée en largeur d'impulsion (MLI ou PWM) et le signal sera crée à partir de l'unité PWM du C167.

L'ensemble de cette application a été mis en œuvre dans un programme **« servo.c »** qui est fourni : il effectue une commande en position du servo-moteur en fonction de la position du potentiomètre. **Cette commande est centralisée et s'exécute donc sur un seul microcontrôleur.**

Un ensemble de fichiers mis à disposition dans */home/MEEA/TPRLI …* est à recopier tel quel sur votre compte :

- **Un répertoire Doc avec 4 fichiers : 2 documentations techniques sur le CAN du C167, une aide pour utiliser la liaison série du C167 et faire de l'affichage sur l'écran du pc : Info_commc167.txt.**
- **servo.c : programme de l'application de commande en position du servo-moteur. Programme complet écrit pour ne tourner sur un seul micro-contrôleur.**
- **un squelette, nommé « squelet….can.c » : il sera à renommer en « can.c » et à compléter**
- **un makefile à modifier/compléter : pas pour de la vrai compilation séparée, juste pour ne pas se tromper et perdre du temps avec la command de compilation**
- **autres fichiers (voir en TP)**

4) TRAVAIL A RÉALISER - PREMIÈRE PARTIE : PRISE EN MAIN DU CONTRÔLEUR DE BUS CAN PRÉSENT SUR LE MICROCONTRÔLEUR 167CR

Il s'agit d'apprendre à utiliser les trames de DONNÉES et les trames de REQUÊTES du protocole CAN. Seulement 2 abonnées sont nécessaires. Il s'agira, de réaliser successivement :

1) l'envoie d'une trame de DONNÉES avec 1, puis 2 caractères (1 caractère = 1 octet) : fonction **CAN_send**
2) la réception de cette trame de DONNÉES avec 1, puis 2 caractères : fonction **CAN_receive**
3) modifier les codes pour envoyer cette fois 1 entier (codé sur 2 octets)
4) la « réponse » à une trame de REQUÊTE : fonction **CAN_setremote**
5) l'envoi d'une trame de REQUÊTE : fonction **CAN_reqremote**

Pour cela :

- Étudier la bibliothèque de fonctions qui vous est fournie (dans le fichier squelette) pour vous aider à programmer le contrôleur. Étudier les prototypes des « primitives » de base du contrôleur CAN auxquelles il faut faire appel : elles sont fournies dans le fichier « ***include/can_16x.h*** » et les détails de leur programmation sont explicités dans le document ***infineon ap292201***
- En se basant sur la bibliothèque ci-dessus, compléter le squelette « …..*c* » qui est donné et qui permettra de disposer de primitives d'émission/réception simples : CAN_send, CAN_receive, ….

5) TRAVAIL A RÉALISER - SECONDE PARTIE : DISTRIBUTION DE L'APPLICATION SERVO.C

L'objectif de cette seconde partie est de distribuer l'application servo.c, c'est-à-dire associer la gestion de chaque élément à un microcontrôleur dédié, et communiquer les informations par le bus CAN. Pour cela, étudier le programme fourni afin d'en dissocier chacune des composantes et d'y insérer ce qui est nécessaire pour assurer les échanges d'information à l'aide de trames CAN, en utilisant les primitives de communications étudiées en partie 1.

A l'aide des fonctions développées dans la partie 1, mettre en œuvre le système de contrôle de la Figure 1. Écrire trois programmes indépendants que à exécuter sur trois microcontrôleurs différents.

Deux types de mise en œuvre seront envisagés :

1) Tous les échanges se feront selon le principe de PRODUCTEUR/CONSOMMATEUR, uniquement à l'aide de trames de DONNÉES. Plus précisément, le capteur émet régulièrement la valeur mesurée du potentiomètre. A la réception de cette valeur, le calculateur (ECU de gestion) calcule la commande pour l'actionneur et émet cette commande. A la réception de cette commande, l'actionneur applique cette commande.
2) Les échanges Capteur-Calculateur se feront en CLIENT/SERVEUR, donc à l'aide de trames de REQUÊTES. Plus précisément, cette fois, le capteur n'émet sa valeur que s'il en reçoit la demande par le calculateur.

6) TRAVAIL A RÉALISER - TROISIÈME PARTIE : … EN COURS DE DÉFINITION

VARIABLES ET PRIMITIVES DE BASE DU CONTRÔLEUR CAN À UTILISER DANS CE TP

a) VARIABLES utiles (2 tableaux) :

- **char dir_bit_16x[16] : valeur des champs (bits) de direction**
- **char dlc_16x[16] : longueur du message émis/reçu**

b) FONCTIONS utiles :

- **init_can_16x(…) : initialisation du module CAN**
- **def_mo_16x (…) : définition/initialisation d'un message objet**
- **ld_modata_16x(…) : charge une donnée dans le registre**
- **send_mo_16x(…) : émet le message**
- **check_mo_16x(…) : vérifie la présence d'un message**
- **rd_modata_16x(…) : copie la donnée d'un message reçu**

ANNEXE 1 - SUR LE CAN DU C167 CR

Fonctionnement schématique du contrôleur de bus CAN du C167 CR

Tout contrôleur de bus CAN à 3 grandes fonctions autonomes (seules les 2 première nous intéressent, la troisième n'étant pas accessible à une gestion par un utilisateur) :

1) Envoyer des trames à partir de messages définis par l'utilisateur (ID_CAN et contenu) ;
2) Recevoir des messages correspondant à un ID_CAN défini par l'utilisateur ;
3) Assurer les fonctions de la norme CAN pour la qualité du trafic : gestion des erreurs, vérification des CRC et accusé de réception pour toutes trames circulant sur le bus.

C'est grâce à l'unité centrale (la CPU) du micro-contrôleur C167, que l'utilisateur gère le contrôleur de bus CAN pour le contrôler, lui envoyer des messages à émettre ou lire des messages reçus.

Envoi d'un message

L'utilisateur, via la CPU, définit le message à envoyer dans un ensemble de registres intitulé ***Message Object*** (**MO** en abrégé dorénavant). Pour des raisons de commodité, nous organisons ces registres en *structure* (dans le sens qu'en donne le langage C).
Le contrôleur CAN du C167CR offre 15 MO, numérotés de 1 à 15. L'utilisateur choisit un MO et doit :

- Placer les données à envoyer (de 0 à 8 octets dans le champ des données constitué de 8 registres, DATA0-DATA7)
- Placer l'ID_CAN complète (11 ou 29 bits) du message dans le champ Arbitrage incorporant un ou deux registres : UAR (*upper arbitration register)* pour un ID de 11 bits (CAN standard), UAR et LAR pour un ID sur 29 bits (CAN étendu). Il existe un registre de masquage *GMS* permettant de configurer le filtrage des messages en réception.
- Décrire les caractéristiques du message dans le registre configuration *MCFG* (*Message Configuration Register*), à savoir : sa longueur (*DLC*, en octets), son format standard ou étendu (*XTD)* et le fait qu'il s'agit d'un message à être émis (DIR=1).
- Pour un envoi commandé, il faut mettre l'indicateur TXRQ (Transmission Request) dans le registre MCR (*Message Control Register).* Quand l'envoi est effectué le contrôleur met à zéro cet indicateur.

Réception d'un message

Pour qu'un MO puisse recevoir une trame de données, il suffit qu'il soit déclaré valide (*MSGVAL=1*) et qu'il soit orienté en réception en fixant *DIR=0*. Si une trame, avec le même ID_CAN que celui définit pour ce MO et le même format que celui consigné dans *XTD*, arrive par le bus CAN, le contrôleur en extrait les informations suivantes qu'il mémorise dans le MO :

- Il recopie l'ID_CAN du message reçu dans le champ Arbitrage
- Il indique dans *DLC* de *MCFG* la taille du message
- Il place les octets du message dans le champ DATA0-DATA7,
- Enfin il signale la réception en mettant l'indicateur *NEWDAT* du registre *MCR* à 1.

ANNEXE 2 - SUR LE MICROCONTRÔLEUR SIEMENS C167

(toutes les informations données dans cette annexe ne sont pas utiles pour le TP

1. Carte microsys167-Eth

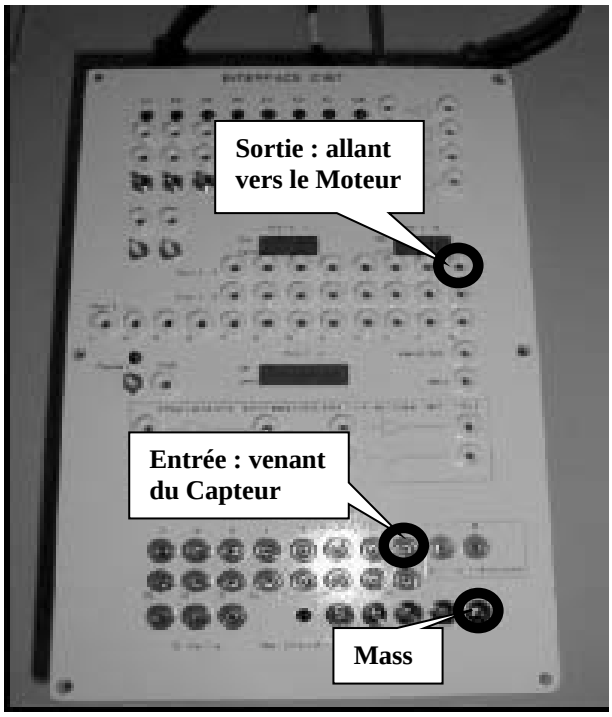
Sur cette carte est conçue autour d'un microcontrôleur Siemens C167 CR cadencé à 20MHz, et dispose en particulier de 1 Moctets de RAM, 512 Koctets de Flash-EPROM et **des interfaces de communication** suivantes :

- une interface série synchrone (RS485),
- **une interface série asynchrone et synchrone (RS232),**
- **une interface bus CAN (Controller Area Network),**
- une **liaison Ethernet** qui sera utilisée pour le téléchargement et la mise au point (débogueur),
- un connecteur externe de 96 points, dont un certain nombre sont ramenés sur la face avant d'un *« boîtier d'interface »* (Ports P7, P3, P2 et P5, reset).

La face avant de ce *« boîtier d'interface »* est illustrée sur la figure ci-après. Les connecteurs des interfaces série, CAN, Ethernet se retrouvent en face arrière du boîtier d'interface.

2. L'interface carte microsys167-Eth <-> extérieur

Pour ce TP, ne sont à manipuler que les 3 points avec une légende sur la figure



Sur la face avant sont disponibles :

- 8 interrupteurs, 2 boutons poussoirs,
- 8 LEDs,
- 4 inverseurs TTL,
- 1 bouton reset, une entrée ~~NMI~~
- 1 horloge de 1 MHz et son entrée de validation (ValidOSC)
- 4 adaptateurs analogiques permettant de ramener une tension [-5V, +5V] en une tension [0V,+5V],
- 16 entrées analogiques correspondant au Port 5,
- les ports d'entrée-sortie logiques Port P7 (huit bits), Port P3 (huit bits), Port P2 (douze bits). Ces ports sont programmables en entrée/sortie mais il faut aussi sur l'interface positionner les lignes en entrée/sortie par des interrupteurs deux positions ON/OFF (de type ``DIL''). Cette contrainte est liée à l'interface, les interrupteurs pilotent des buffers de bus placés sur les lignes des ports. 2 sorties analogiques 0 et 1, sélectionnées par Chip Select et correspondant respectivement aux adresses 0x2xxxxx et 0x3xxxxx (voir polycopié de cours, chapitre 3).

3. Compilation et édition de liens

La compilation d'un programme C en vue de la génération d'un code exécutable destiné à une cible telle que le C167, fait appel à un compilateur (avec des options) dédiés à cette cible.
A la base, c'est le compilateur gcc166, invoqué par la commande : ***gcc166 -m7 -g -o prog prog.c*** (**m7** précise que la compilation doit se faire pour une cible C167).

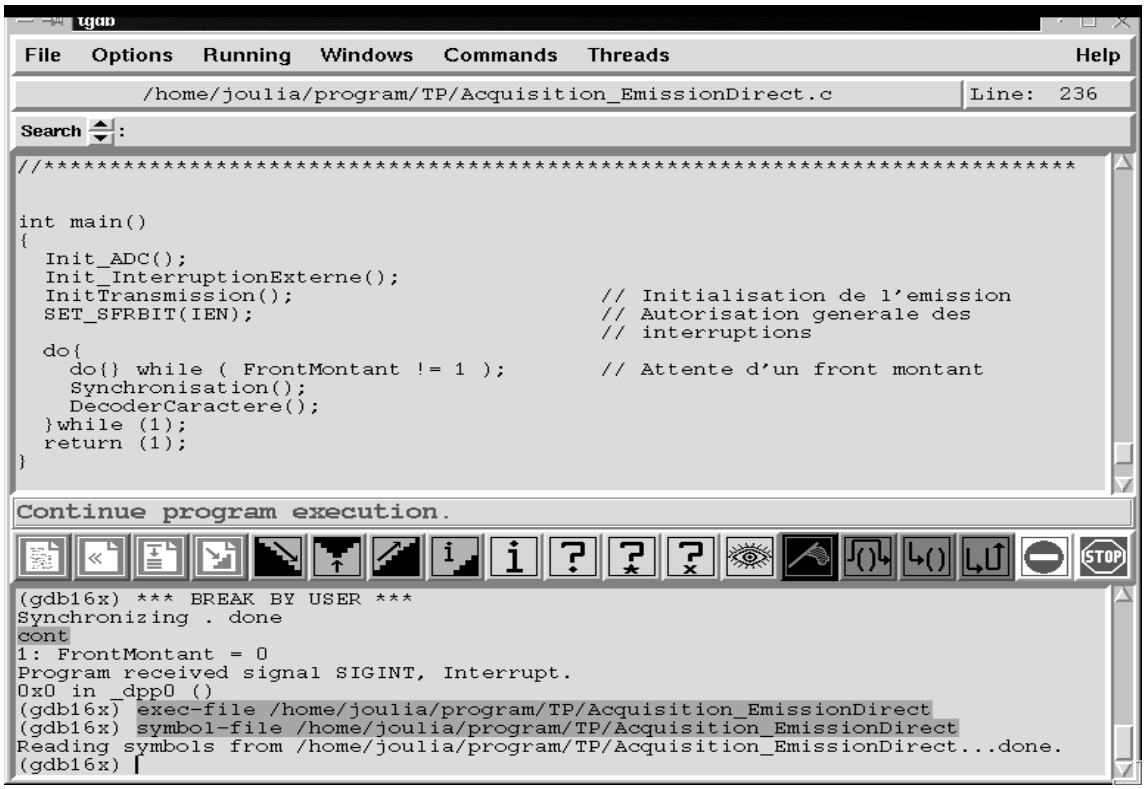
MAIS, DANS CE TP, il faut utiliser la commande présente dans le fichier « makefile » fourni, pour plus de facilité, et à cause des besoins plus spécifiques pour le contrôleur de bus CAN.

4. Environnement logiciel / Téléchargement / Mise au point

Dans le principe, dans l'installation faite en salle de TP, n'importe quelle carte microcontrôleur peut être « téléchargée » à partir de n'importe lequel des **PC linux** via une **liaison ethernet. Les PC et les cartes ont un nom** (représentatif de leur adresse IP)**.**
Pour télécharger un programme sur une carte à partir d'un PC, il faut donner le nom de carte. Ce nom a le format : *mX*, avec X le nom du PC « à coté » de la carte, ou dit autrement, le PC auquel le microcontrôleur est relié par sa liaison série (voir exemple plus loin).

Le chargement de l'exécutable sur la cible et la mise au point à distance se font par l'intermédiaire du débogueur (tgdb166, adaptation du débogueur gdb).

1) Lancement du débogueur par la commande : ***tgdb166***
2) Une fenêtre débogueur s'ouvre (voir figure suivante), qui présente une barre de menu (File, Options, Running, ...) et les trois fenêtres classiques :
   - fenêtre source,
   - bandeau de commande : présentation de commandes *xxgdb* sous forme de boutons,
   - fenêtre de travail : entrée des commandes et affichage des résultats des commandes.



3) La connexion du débogueur au moniteur de la carte cible nommée ***mX*** est obtenue par la commande *target* : ***target c16x mX.ups-tlse.fr$10200***.

Par exemple, sur carte associé à la machine tamia : *target c16x mtamia.ups-tlse.fr$10200,* avec *:*

- ***c16x*** désigne le type de carte(C167),
- ***mX*** le nom de la carte, où X est le nom du PC (qui n'a pas besoin d'être alumé) auquel le microcontrôleur est relié par sa liaison série
- ***$10200*** est un service réseau pour le téléchargement.

5. Mise au point, le débogueur

Une session de travail type (sur *tamia* et la carte *mtamia*) se présente ainsi :

- *édition du fichier source*
- *compilation : voir le makefile fourni*
- *lancement du débogueur : tgdb166*
- *choix du fichier à télécharger : Menu file, puis program, et sélectionner le nom du fichier exécutable désiré dans la liste des fichiers présentés,*
- *connexion à carte cible: target c16x mtamia.ups-tlse.fr$10200*
- *après demande de confirmation du chargement, et un reset éventuel de la carte cible, le programme sera chargé.*
- *ce programme doit apparaître dans la fenêtre source, sinon repasser dans le menu file, puis visit et choisir le programme désiré dans sa forme en code source (prog.c)*
- *le programme peut être lancé (symbole ``drapeau'' du bandeau de commande)*
- *ou exécuté pas à pas (sans entrer dans les fonctions ou bien en pas à pas dans les fonctions, boutons correspondant respectivement aux deux symboles à droite du drapeau),*
- *des points d'arrêt sont placés et supprimés sur un clic du bouton droit de la souris,*
- *des variables peuvent être visualisées en les sélectionnant et cliquant sur le bouton ``?''*
- *des variables peuvent être suivies lors de la mise au point (fonction display de xxgdb) dans une fenêtre d'affichage (menu Windows puis Watches). La sélection des variables se fait par sélection avec la souris, puis clic sur le symbole ``oeil''.*