

# Systèmes interconnectés – SINT

## Sujet de TP 4 : CAN

Centrale Nantes

P.-E. Hladik, pehladik@ec-nantes.fr

---

Version bêta (20 novembre 2023)

### 1 Objectifs pédagogiques

- mettre en œuvre un échange de messages sur un bus CAN
- observer et comprendre le bit stuffing et l'arbitrage de priorité
- estimer les temps de transition d'un message CAN

### 2 Rendu

Le travail est prévu pour deux séances de TP. À la fin, déposez sur Hippocampus une archive avec un (petit) compte rendu en pdf et les extraits de code pertinents.

### 3 Mise en œuvre d'une communication sur un bus CAN

Vous allez utiliser les cartes Feather M0 et Teensys pour mettre en œuvre une communication sur le bus CAN à l'aide des bibliothèques Arduino.

#### 3.1 Envoi et réception d'un message sur le CAN

Le premier objectif est de faire changer l'état de la led  $L[x]$  ( $x=0..2$ ) de la carte Teensy 3.6 suite à l'appuie sur le bouton poussoir  $P[x]$  ( $x=0..2$ ) de la carte Feather M0 (voir figure 1). Le schematic de la carte teensys peut être trouvé sur [schematics.pdf](#). Mais d'ailleurs sur quelle broche est connecté le CAN ?

**Installation des librairies.** Pour ce TP vous utiliserez les libraires ACAN pour le Teensys et ACAN2515 pour la Feather.

Commencez par télécharger la dernière version de la librairie ACAN à partir de l'environnement Arduino (menu Croquis > Inclure une bibliothèque > Gérer les bibliothèques). Faites de même pour la librairie ACAN2515.

Les bibliothèques ACAN et ACAN2515 contiennent les fichiers source qui seront compilés, la documentation en PDF (à consulter pour faire ce TP) et des croquis d'exemple, dont LoopBackDemo et LoopBackDemoAdafruitFeatherM0 qui peuvent vous servir de base comme exemple (les versions avec filtre peuvent être aussi utiles).

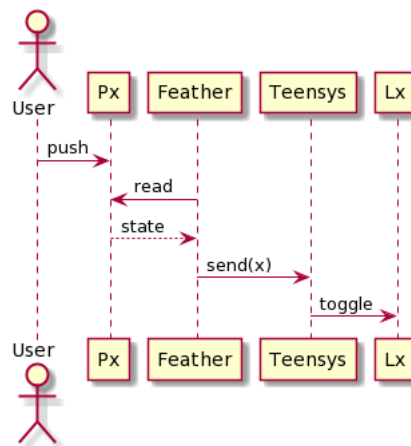


FIG. 1 – Diagramme de séquence pour le contrôle des leds

Pour ouvrir les croquis d'exemple LoopBackDemo et LoopBackDemoAdafruitFeatherM0 aller dans le menu Fichier > Exemples > ACAN > LoopBackDemo ou Fichier > Exemples > ACAN2515 > LoopBackDemoAdafruitFeatherM0.

**Test et mise au point.** Afin de mettre au point votre application, vous pouvez facilement réaliser des tests en mode LoopBack.

**Contraintes.** Vous devez utiliser les filtres de réception.

**Conception.** C'est à vous de définir les messages : identificateur, structuration de l'information, etc. Indiquez dans le rapport le format des vos messages.

**Conseils.** Vérifier que les interrupteurs de terminaison sont activés sur les interfaces.

### 3.2 Mise en place d'une communication de type requête-réponse

Le second travail consistera à mettre en place un système pour afficher sur la Teensys l'état du codeur incrémental de la Feather suite à l'appui du boutons P0 sur la Teensys. Proposer une solution avec un mécanisme de type *Request-reply* (figure 2a) et un second avec une publication périodique de l'état du codeur incrémental (figure 2b).

Pour chacune des solutions, mesurer le temps de réaction entre l'appui sur le bouton et l'affichage, ainsi que la fraîcheur de la valeur. Pour faire ces mesures, programmer des broches sur les cartes pour générer des signaux permettant de mesurer les temps et utiliser l'analyseur pour relever ces informations.

Si on voulait faire un affichage périodique de la valeur du codeur de la Feather sur la Teensys quelle solution vous semblerait la plus appropriée ? Expliquez dans le rapport pourquoi.

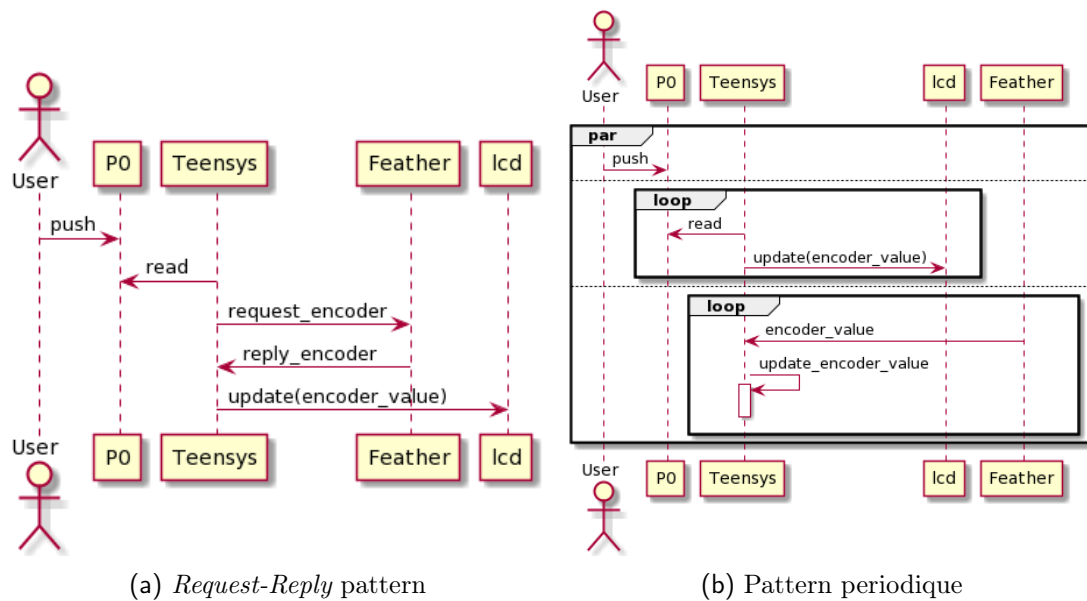


FIG. 2 – Patterns d'échange de données

## 4 Bit-stuffing, arbitrage et détection d'erreur

Connecter la Teensy 3.6 à la STM32. La liaison CAN se fera entre le CAN0 de la Teensy 3.6 et la carte STM32 (qui n'a qu'un module CAN). Vous avez les broches TX et RX du CAN de la STM32 (entre le transceiver et le  $\mu C$ ) qui sont accessibles pour connecter un analyseur logique (le schéma de la petite carte STM32 est sur Hippocampus).

### 4.1 Observation du bit-stuffing

Envoyer une trame de données standard, dont l'identificateur est 0x110, qui contient 2 octets de données (0x10 et 0x34 par exemple). La carte STM32 doit répondre avec une autre trame de donnée.

Quelle est l'identifiant de cette trame? Comment ont été modifiées les données (il faudra certainement faire plusieurs essais avec des données différentes)? Expliquer les différences entre les broches RX et TX (lors de l'envoi et de la réception d'une trame par le nœud STM32). Pour les données 0x10 et 0x34, combien de bits sont rajoutés (bit stuffing)?

### 4.2 Identification d'une trame

Il y a 4 interrupteurs et 2 boutons poussoirs sur la carte STM32. On n'utilisera pas encore le bouton poussoir (PA1) le plus proche du connecteur CAN! On sait que la modification d'un des 4 interrupteurs, ou l'appui sur le bouton poussoir (PA0) génère l'envoi d'une trame standard de 1 octet, mais l'identifiant a été perdu! On sait uniquement qu'il se situe entre 0x200 et 0x300.

Déterminer l'identifiant associé et le protocole utilisé (même principe que pour la partie liaison série). Programmer une application basique tel que chaque interrupteur allume une led différente de la teensy3.6

### 4.3 Identification d'une trame (bis)

Il y a 4 leds sur la carte STM32. On sait que ces leds sont pilotées par une trame standard de 1 octet, mais l'identifiant a aussi été perdu ! On sait uniquement qu'il se situe entre 0x10 et 0x100.

Déterminer l'identifiant associé et le protocole utilisé (même principe que pour la partie liaison série). Les LEDs L0 à L3 sont respectivement connectées sur les broches D12, D11, D3 et D6 de la carte STM32.

### 4.4 Trame d'erreur

L'appui sur le bouton poussoir (PA1) à côté du connecteur CAN provoque l'envoi d'une trame avec une erreur sur le bus. Pour forcer cette erreur, le nœud commence une transmission CAN, puis passé un court délai reconfigure la broche de transmission en I/O sur un court laps de temps pour générer une erreur. La broche D9 est mise à 1 pendant ce laps de temps.

Expliquer le comportement (analyseur logique) : Quel est le déroulement chronologique ? Qu'est ce qui a été modifié sur la trame ? Quelle est l'erreur associée ? Comment réagissent les autres nœuds ? Quel est la durée totale de cette gestion d'erreur sur le bus ?

Attention : le décodeur de protocole n'est pas capable de détecter une erreur (fonction non implémentée). Il sera nécessaire de regarder directement la trame binaire !

## 5 Application à trois partenaires

Repartez du code que vous avez produit dans la partie 3.1 et connectez la STM32 sur le bus.

Concevez un système tel que :

- la carte Feather M0 WiFi envoie périodiquement (5 ms) une trame qui contient l'état du poussoir P0, puis une contenant l'état de P1, puis de P2 et enfin la valeur de l'encodeur (les 4 bits de poids faibles),
- la carte Teensy 3.6 allume et éteint les leds L0 à L2 en fonction de l'état des poussoir P0 à P2,
- la carte STM32 allume ses leds en fonction de la trame représentant la valeur de l'encodeur (voir les questions précédentes). Les Leds de la STM32 reflèteront ainsi la valeur de l'encodeur.