

Programme de tso

**PROJET FIN DE SESSION 5**

**CAN bus**

**Travail présenté**

**À**

**Stéphane Deschênes**

**Kevin Cotton**

**Yves Roy**

**Par**

**Samuel Duclos**

**Cégep Limoilou**

**Québec, 2019-12-12**

Table des matières

[Fonctionnement général : 3](#_Toc27065997)

[Schéma bloc : 3](#_Toc27065998)

[Schéma de câblage : 3](#_Toc27065999)

[Tableau descriptif du câblage : 4](#_Toc27066000)

[Explication du fonctionnement du P.C. : 4](#_Toc27066001)

[Schéma du fonctionnement du P.C. : 4](#_Toc27066002)

[Explication du fonctionnement du premier BeagleBone Blue : 5](#_Toc27066003)

[Schéma du fonctionnement du premier BeagleBone Blue : 5](#_Toc27066004)

[Explication du fonctionnement du STM32F407VGT6 Discovery Board : 6](#_Toc27066005)

[Schéma du fonctionnement du STM32F407VGT6 Discovery Board : 6](#_Toc27066006)

[Explication du fonctionnement du deuxième BeagleBone Blue : 7](#_Toc27066007)

[Schéma du fonctionnement du deuxième BeagleBone Blue : 7](#_Toc27066008)

[Résumé de ce qui a fonctionné et ce qui n’a pas fonctionné : 8](#_Toc27066009)

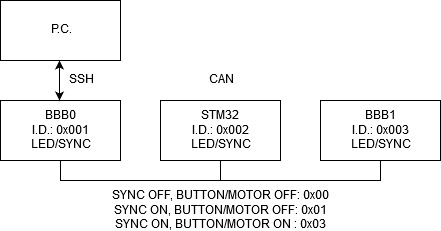
[Ce qui a fonctionné : 8](#_Toc27066010)

[Ce qui n’a pas fonctionné : 8](#_Toc27066011)

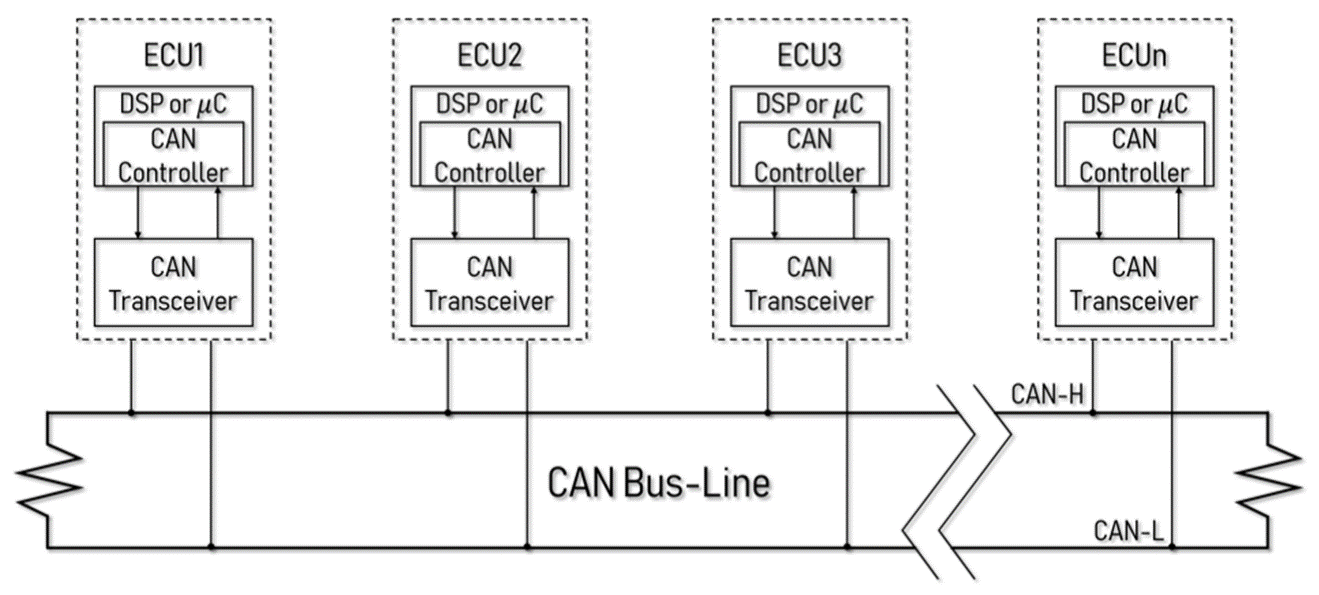
# Fonctionnement général :

La communication entre un P.C. par SSH avec un BeagleBone Blue est commandée en CAN bus par broadcast manager (BCM) cyclique du premier BeagleBone. Le réseau CAN de ce premier BeagleBone est attaché à un STM32F407VGT6 Discovery Board et à un second BeagleBone Blue. Le P.C. envoie un signal de synchronisation au premier BeagleBone qui active celui-ci qui relaie les informations. Le STM32 répond ensuite et vérifie par la trame CAN s’ìl doit activer une LED. Le second BeagleBone lit ensuite un bouton qui active la commande de la LED et le cycle recommence.

# Schéma bloc :



# Schéma de câblage :



# Tableau descriptif du câblage :

|  |  |  |  |
| --- | --- | --- | --- |
| **Rôle** | **Couleurs** | **Origine** | **Destination** |
| CAN JST -> JP1-3 (RJ-45) | Jaune (JST) –> Brun (RJ-45) | BeagleBone | STM32F407VGT6 |
| CAN JST -> JP1-3 (RJ-45) | Bleu (JST) –> Brun/Blanc (RJ-45) | BeagleBone | STM32F407VGT6 |

# Explication du fonctionnement du P.C. :

L’application du P.C. est écrite en C# et interface un terminal SSH. Des librairies sont réutilisées pour simplifier l’implémentation de la solution. Il y a un seul ajout qui a été fait au programme pour permettre la connexion au compte « root » et ainsi être en mesure d’exécuter le programme qui nécessite des permissions spéciales.

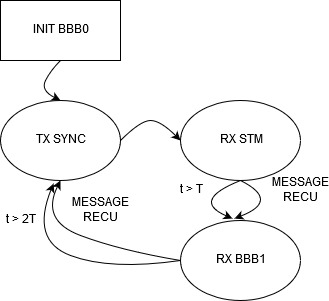
# Schéma du fonctionnement du P.C. :

U:\FRÉDÉRIC DION\Projet\Projet 5e session\Documentation\Schémas\poste_commande.jpg

# Explication du fonctionnement du premier BeagleBone Blue :

Le premier BeagleBone est modulaire dans le sens que les variables modifiables sont déclarées au-dessus du reste du code. Le code du deuxième BeagleBone est presque pareil sauf dans la boucle principale. L’exécution débute par une phase d’initialisation pour éviter de reconfigurer chaque BeagleBone. Cette initialisation commence par charger les modules CAN du noyau Linux. Ensuite, la LED rouge et la LED verte sont configurées en mode persistance. La LED rouge est éteinte et la LED verte est allumée pour indiquer le début du fonctionnement normal du programme. La possibilité de jouer directement avec les « sockets » étant désactivée dans le système de fichiers et ne voulant pas réinstaller des dépendances externes sur chaque BeagleBone, un « character device » est créé pour manipuler les sockets directement s’il n’existe pas déjà. Le programme en profite pour « flusher » les tables de règles du firewall pour permettre de « bind » plusieurs « listeners » sur le même socket. Si l’interface CAN choisie est virtuelle, il n’est pas nécessaire de spécifier le bit rate. L’interface réseau est ensuite créée puis activée. S’ensuit la mise en marche du serveur du broadcast manager CAN qui communique par sockets. Le PID du serveur lancé en « background » est sauvegardé dans une variable pour la phase de destruction de la fin et un délai permet de s’assurer que celui-ci a le temps de bien démarrer. Afin de pouvoir communiquer avec le serveur directement dans le TTY, un descripteur de fichier lie le TTY en entrée et en sortie standard. Un « pipe » serait sous-efficace et contiendrait quand-même des descripteurs de fichiers. Juste après la phase d’initialisation, une fonction est configurée pour attraper les signaux d’interruption « SIGTERM », « SIGINT » et « SIGHUP ». Elle permet le « clean exit » en cas de fermeture par CTRL-C ou autre. Cette fonction envoie le signal de « power down » au prochain cycle, attend ce cycle, tue le serveur, ferme le descripteur de fichier, désactive l’interface réseau, l’efface, puis éteint la LED verte et allume la LED rouge. Le vrai début du programme commence avec la configuration du signal cyclique et la configuration de récepteurs CAN. S’ensuit la boucle qui débute en lisant les messages CAN par le descripteur de fichier et imprime le résultat s’il a changé. Ensuite, si le STM32 envoie un message différent, il le répète. Cela permet d’éviter de configurer de multiples filtres sur le STM32. Le « input » SSH du P.C. est lu avec un timeout qui prend ensuite une décision basée sur le message reçu ou non dans le temps. Si le message venant du P.C. est « ON », la boucle recommence. Si le délai de lecture est dépassé, s’il reçoit la commande OFF ou si la commande reçue est dysfonctionnelle, le programme sort de la boucle et entre dans la fonction « destruction » avant de redonner le contrôle au noyau Linux.

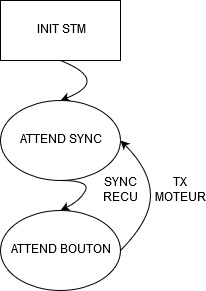
# Schéma du fonctionnement du premier BeagleBone Blue :



# Explication du fonctionnement du STM32F407VGT6 Discovery Board :

La grosse partie des « drivers » est générée par le STM32CubeMX. Le programme fonctionne sur une base de temps en machine d’état en utilisant un tableau de pointeurs de fonctions. Il y a deux phases : la partie qui traite les éléments du protocole au début, puis la partie qui transmet, reçoit et applique les décisions en fonction des entrées. La structure du programme est encore en état de développement mais beaucoup d’éléments du programme de laboratoire original ont été retirés.

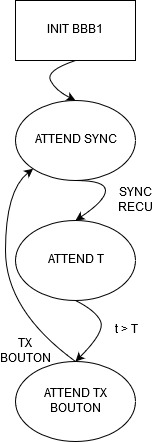
# Schéma du fonctionnement du STM32F407VGT6 Discovery Board :



# Explication du fonctionnement du deuxième BeagleBone Blue :

La première différence significative entre le premier BeagleBone et le deuxième est l’utilisation des interruptions systèmes logicielles du noyau Linux. En effet, le bouton en entrée GPIO est configuré pour déclencher des interruptions système lors d’un front montant. Le reste changeant du programme ne fait que modifier l’état gardé du bouton et en envoyer le résultat par le bus CAN après un délai lors de la réception, si l’interruption a été reçue. Sinon, le message habituel est envoyé périodiquement.

# Schéma du fonctionnement du deuxième BeagleBone Blue :



# Résumé de ce qui a fonctionné et ce qui n’a pas fonctionné :

## Ce qui a fonctionné :

Du côté des deux BeagleBone Blue, tout fonctionne. La boucle principale du deuxième BeagleBone devrait être retravaillée pour éviter le crash lors d’un « CTRL-C », mais sinon, tout est « clean » à part les délais explicites qui étaient encore en développement avant la présentation. Du côté du STM32, il transmet et semblait recevoir jusqu’à il y a quelques « commits ». Du côté de l’application C# du P.C., le feedback fonctionne mais la synchronisation ne se fait pas dans l’interface graphique.

## Ce qui n’a pas fonctionné :

Du côté du BeagleBone Blue qui sert de « input » au bouton, le programme « crash » lorsqu’il est fermé par CTRL-C. Il faut alors fermer le terminal et redémarrer. Pour le P.C., l’interface graphique est très sommaire et la synchronisation ne s’envoie pas à travers l’interface en C#; par contre elle s’envoie dans un terminal comme Bash ou DOS. Du côté du STM32, un bug s’est introduit dernièrement qui n’attend pas la synchronisation pour transmettre en CAN périodiquement et la LED ne s’allume pas.